Reading and Reasoning
with Knowledge Graphs

Matthew Gardner

CMU-LTI-15-014

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Tom Mitchell, Chair
William Cohen
Christos Faloutsos
Antoine Bordes

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies*

Keywords: Knowledge base inference, machine reading, graph inference

To Russell Ball and John Hale Gardner, whose lives instilled in me the desire to pursue a PhD.

Abstract

Much attention has recently been given to the creation of large knowledge bases that contain millions of facts about people, things, and places in the world. These knowledge bases have proven to be incredibly useful for enriching search results, answering factoid questions, and training semantic parsers and relation extractors. The way the knowledge base is actually used in these systems, however, is somewhat shallow—they are treated most often as simple lookup tables, a place to find a factoid answer given a structured query, or to determine whether a sentence should be a positive or negative training example for a relation extraction model. Very little is done in the way of reasoning with these knowledge bases or using them to improve machine reading. This is because typical probabilistic reasoning systems do not scale well to collections of facts as large as modern knowledge bases, and because it is difficult to incorporate information from a knowledge base into typical natural language processing models.

In this thesis we present methods for reasoning over very large knowledge bases, and we show how to apply these methods to models of machine reading. The approaches we present view the knowledge base as a graph and extract characteristics of that graph to construct a feature matrix for use in machine learning models. The graph characteristics that we extract correspond to Horn clauses and other logic statements over knowledge base predicates and entities, and thus our methods have strong ties to prior work on logical inference. We show through experiments in knowledge base completion, relation extraction, and question answering that our methods can successfully incorporate knowledge base information into machine learning models of natural language.

Acknowledgments

This thesis was not produced in a vacuum, and there are many people whose ideas and encouragement contributed to the work described here. My advisor, Tom Mitchell, provided crucial support and guidance throughout the course of my PhD, most importantly providing a research environment and set of interesting questions where this work could thrive.

I am most indebted to Partha Talukdar for the particular direction that this thesis took; while Tom first nudged me to consider working on PRA, it was Partha whose ideas formed the foundation of my first two papers on the subject. I am indebted as well to Dheeru Dua, who actually ran most of the experiments that went into Chapter 4 of this thesis.

I also had many positive interactions with the rest of Tom's research group that helped solidify my thinking on many topics related to this thesis. Justin Betteridge, Derry Wijaya, Burr Settles, Ndapa Nakashole, Alan Ritter, Bryan Kisiel, Anthony Platanios, Abulhair Saparov, and especially Jayant Krishnamurthy have been valued colleagues and friends during my time at Carnegie Mellon University.

Lastly, I thank my family. My parents Robert and Debbie Gardner helped shaped me into a person capable of finishing a PhD, and my wife Sabrina and my children Beth, Russell and David have been my motivation, my refuge, and my joy. I could not have done this without them.

Contents

List of Figures

List of Tables

List of symbols and abbreviations

KB	Knowledge base (see Section 2.1)
PRA	The path ranking algorithm (see Section 2.2.2)
SFE	Subgraph feature extraction (see Section 3.4)
SVO	Subject-verb phrase-object; a triple extracted from a syntactic analysis of a sentence, such as (horses, can eat, hay).
NELL	Never Ending Language Learner, an automatically constructed knowledge base used for several experiments in this thesis.
Freebase	A knowledge base constructed by hand by many contributors, used in several experiments in this thesis.
ClueWeb	A large collection of web documents (Callan et al., 2009), used as the source of the SVO triples used in many of the experiments in this thesis.
GraphChi	A library for performing efficient graph computations, developed by Aapo Kyrola and used to implement PRA for the experiments used in this thesis.
(e_s, r, e_t)	A triple from a knowledge base, representing the fact that a relationship r holds between a source entity e_s and a target entity e_t.
FORMALKBRELATION	A relation or entity from a formal knowledge base is formatted in small caps.
“surface relation”	A relation or noun phrase extracted from text is surrounded by quotes.
$\mathcal{G}$	Generally denotes a graph.
$\mathcal{N}$	Generally denotes the nodes in a graph.
$\mathcal{E}$	Generally denotes edges a graph.
$\mathcal{R}$	Generally denotes the set of edge labels (or relations) in a graph.
π	Used to denote a *path type*, or sequence of edge labels, in a graph.
l	Used to denote the *length* of a path type π.
α	The restart probability defined in Section 3.3.3.
β	The spikiness parameter defined in Section 3.3.3.
BFS	Breadth-first search
PPR	Personalized page rank

Chapter 1

Introduction

Much work has recently been put into the construction of very large knowledge bases. Knowledge bases (KBs) are collections of facts, about people, things, and places in the world, and relationships between them. They can be predominantly focused on proper entities (people and places), as Freebase is (Bollacker et al., 2008), or on common-sense knowledge, like ConceptNet (Liu and Singh, 2004). Some knowledge bases define an ontology, a set of allowable categories and relations for entities in the knowledge base (e.g., Freebase and NELL (Carlson et al., 2010)), and other knowledge bases are "ontology free", allowing free text to define the categories and relations that exist in the KB (Etzioni et al., 2011). While some knowledge bases are created manually, a lot of recent research has gone into creating them automatically, with entire workshops devoted to the subject (Suchanek et al., 2013).

These knowledge bases can be used for a variety of tasks: search engines now use them to enrich search results with structured information; many question answering systems convert a question into some kind of query against a knowledge base; and information extraction systems will often use a knowledge base as their main source of training data. However, even the largest knowledge bases are woefully incomplete, limiting the performance of systems that rely on the information contained in the knowledge base. One way this incompleteness is manifest is in the percentage of encodable facts that are missing from the knowledge base. For example, the knowledge base may try to capture information about basketball teams and their players, but no knowledge base has complete information about all basketball games ever played, especially considering that people are still playing basketball. Another way in which all (formal) KBs are incomplete is that the vast majority of facts about the world are not encodable by the predicates in the KB at all—Freebase encodes information about which players play for which teams, but not about how many points each player scored in a particular game.

Recent work has attempted to overcome the first of these problems by reasoning over the facts contained in the knowledge base to infer facts that are missing, a task often called *knowledge base completion* or *knowledge base inference*. Approaches to the task of knowledge base completion can be broadly placed into two main categories: methods that use some kind of logical inference to predict missing facts, and methods that find a vector space model of predicates and entities in the knowledge base and use this to infer new facts. For example, a method using logical inference might note that Michael Jordan is teammates with Scottie Pippen, and that Pippen plays for the Chicago Bulls, and from this conclude that Michael Jordan also plays for the Bulls. In contrast, a vector space method might find that the vectors for Michael Jordan and Scottie Pippen are similar, so they should have similar values for which team they play for. The methods performing logical reasoning are generally more precise and can make more powerful inferences than vector space models, but they also tend to be more brittle and can break down when there are a large number of predicates or facts to reason over.

The second aspect of the incompleteness of KBs, that of missing *predicates* in the KB, is much more challenging. As it is not feasible to encode all possible predicates in a KB, there will always be facts expressible in natural language that cannot be found in a formal knowledge base. One approach to bypass this issue is called “open” or “ontology-free” information extraction (Etzioni et al., 2011). Instead of defining a formal predicate space and populating a knowledge base, open information extraction techniques seek to find relationships expressed in natural language text, and use the words in the text to define their predicate space. The main drawback of this approach is that reasoning is much more difficult without a set of formal predicates; there are many ways to say very similar things in natural language, and typical reasoning systems treat all of these ways as independent predicates, making generalization across similar predicates challenging.

This work takes steps toward solving both of these problems, introducing methods for improved *reasoning* over large formal knowledge bases, and applying these methods to simple models of *reading* open-domain text. The thesis of this research is that feature extraction from graphs is a scalable and effective means of incorporating knowledge base information into machine learning models of reasoning and reading.

We begin by considering the task of reasoning with a knowledge base to predict missing facts. The methods for knowledge base inference that we present in this thesis build off of a prior technique called the path ranking algorithm (PRA) (Lao and Cohen, 2010b). PRA views the knowledge base as a graph, then performs random walks over the graph to find sequences of relations (or paths) that are potentially predictive of new instances of a particular relation. PRA then uses these paths as features in a logistic regression model for inferring missing facts in the KB. This is akin to Horn clause learning and has strong connections to logical inference. This method is *compositional*, in that it can make use of compositions of relations as features to predict new facts, and as such it has significantly more representational power than many other modern techniques for KB inference. Its other main advantage is that random walks scale easily to very large graphs and allow for any-time inference; more or fewer walks can be performed as time for inference permits. The representational power of Horn clauses gives the method its main drawback, however: the feature space considered by PRA is very large and very sparse, and it lacks a mechanism for exploiting similarity among the predicates in the KB to decrease the effective size of the feature space.

We present three successive methods that improve on the reasoning done by PRA, ultimately improving mean average precision by 46% while simultaneously decreasing running time by an order of magnitude. The first two methods we present modify PRA to make better use of vector space representations of relations when they are available. This technique allows PRA to overcome the sparisity inherent in very large predicate spaces with a lot of synonymy. The third method we introduce is a simplification of the path ranking algorithm, removing steps that are computationally expensive but provide no observable benefit on the tasks we consider in this thesis. Instead of computing probabilities with random walks, this method, which we call subgraph feature extraction (SFE), first finds a subgraph around a pair of nodes with a simple breadth-first search, then extracts features from that subgraph. This simpler algorithm allows us to construct much more expressive feature matrices in a fraction of the time, leading to significantly better performance on knowledge base inference.

While we present these methods in the context of the task of knowledge base inference, at a fundamental level these methods *generate feature matrices* over node pairs in a graph. When performing knowledge base completion, these feature matrices can be used in a simple logistic regression model to predict whether an edge of a particular label should exist between a pair of nodes. But that is not the only possible application of these feature matrices. The remainder of this thesis turns from the task of knowledge base completion to the problem of reading open domain text. We first show that the feature matrices generated by PRA and SFE can be used to incorporate background knowledge into the task of relation extraction, predicting which formal KB relation is expressed by a particular natural language sentence, if any. We then conclude by demonstrating that these reasoning methods can also successfully model lexical semantics; that is, in addition to reasoning over formal predicates in a semantically-limited knowledge base, these methods can capture (at least some of) the meaning of *actual language* used in context. This ability to model lexical semantics is a core component of any open-domain reading or question answering system.

1.1 Chapter Overview

This thesis is organized as follows. Chapter 2 situates this thesis in the context of prior work. It first gives a more formal description of what is meant by "knowledge base" in this thesis. It then describes prior methods for reasoning with knowledge bases, including giving a detailed description of the path ranking algorithm, as PRA is the foundation for much of the contributions in this work. The chapter concludes with a brief mention of other related work, focusing largely on relation extraction.

Chapters 3 through 5 contain the novel contributions of this thesis. Chapter 3 introduces our methods for reasoning over knowledge bases. In three main sections, we present three successive improvements over the original PRA algorithm, evaluating performance on the task of knowledge base inference. Following this, Chapter 4 shows how these techniques can be applied to the task of relation extraction, and Chapter 5 shows that our graph-based reasoning techniques can also learn models of open-domain lexical semantics, evaluating them on two separate question answering tasks.

In Chapter 6 we conclude, summarizing what we have learned and offering suggestions for future work.

Chapter 2

Background

This chapter provides background information on the methods used in this thesis, and situates the main contributions we present in the context of prior work. Section 2.1 formally defines the concept of a knowledge base and discusses several representational decisions that must be made when using a knowledge base in a machine learning algorithm.

Section 2.2 gives an overview of prior methods for reasoning with knowledge bases. We first discuss methods that are based on logical inference. These methods are highly expressive and powerful, but they typically do not scale well to the large size of modern knowledge bases. Graph-based methods such as the path ranking algorithm (PRA), which we describe next, aim to use similar information to that available to logic systems, but in a way that scales better to large knowledge bases. The last class of KB inference techniques we describe are those that use some kind of vector space embedding of the KB in order to predict missing facts. These methods cannot perform the complex reasoning available to the other systems, but they are more robust to noise and sparse training data. The work in this thesis primarily builds off of the graph-based approaches, both making them more efficient and incorporating information from vector space models of KB inference.

Finally, Section 2.3 describes other work that is related to this thesis, focusing primarily on relation extraction.

2.1 What is a knowledge base?

The main task this thesis focuses on is knowledge base completion. To discuss this task, we must first define what we mean by a knowledge base. The term *knowledge base* is most frequently used with large collections of curated knowledge, such as Freebase (Bollacker et al., 2008), YAGO (Suchanek et al., 2007), or CYC (Lenat, 1995), and it is also applied when this knowledge is automatically constructed, such as with NELL (Carlson et al., 2010). Here, we use it to refer to *any* collection of facts, no matter their source or underlying ontology, if any. This can include subject-verb-object triples automatically extracted from large text corpora, for example, and other such open-domain relations obtained with an open information extraction technique such as ReVerb or OLLIE (Fader et al., 2011; Mausam et al., 2012).

We formally define a knowledge base as a collection of triples, (e_s, r, e_t), where each triple expresses some relation r between a source entity e_s and a target entity e_t. As stated above, the relations r could be from an underlying ontology (such as /TYPE/OBJECT/TYPE, from Freebase, or CONCEPT:LOCATEDAT, from NELL), or they could be verb phrases extracted from text, such as "plays for". The entities e could be formal representations of real-world people, places, categories, or things (such as /M/02MJMR from Freebase or CONCEPT:BARACKOBAMA from NELL, both representing U.S. President Barack Obama), or they could be noun phrases taken directly from text, such as the string "Obama". To represent relations with more than two arguments (an *n-ary relation*), an entity e could also represent a reified relation instance (called a mediator in Freebase), where each of the n arguments in the n-ary relation shares a triple with the reified relation instance. For example, Freebase has a reified relation type called /FILM/PERFORMANCE, which is used to relate a film, an actor, a character, and a performance type. Film performance entities in Freebase are simply abstract concepts that exist only to connect more than two entities into a single relation.

2.1.1 Combining textual and formal KBs

An important question arises when trying to make use of triples extracted from text along with triples in a curated knowledge base. The entity spaces of the two sets of triples are disjoint (one uses strings, and the other uses formal representations), and so without combining or linking them in some way it is impossible for information to flow from one set of triples to the other. There are several options for how to combine these two sets of triples, and the decision has important ramifications on both the power and the scalability of inference.

The first (and most popular) option is to discard one of the entity spaces after performing some kind of entity linking between the strings and the formal representations. Thus "Obama" might be linked to the formal entity CONCEPT:BARACKOBAMA, either by string matching or by using the context of the document, and whatever relations were participated in by the entity "Obama" are transferred to the formal entity. This process is often noisy and is always very incomplete, as there will be many noun phrases seen in text that cannot be mapped to a formal entity representation, and triples containing these entities are generally just discarded. As a simple toy example, consider the small graph in Figure 2.1. The noun phrases "J. K. Rowling" and "Harry Potter" have alias links to entities in the KB, but "wizards" does not. Discarding all triples involving noun phrases not linked to KB entities (such as those involving "wizards") would remove the connection shown in the figure.

An alternative approach is to maintain two distinct entity spaces, but add triples to link them. These triples have the relation ALIAS, or CANREFERTO, meaning that the noun phrase can be used to reference the formal entity. This is a much weaker connection than using an entity linking system to map a noun phrase in context to a formal entity, but it also avoids the need for an expensive (and potentially error-prone and lossy) preprocessing step over the corpus. Another important point is that the addition of a connecting relation such as this is only helpful for methods that can make use of longer-range dependencies than a single matrix row—the factorization methods we describe in Section 2.2.3 can produce low-dimensional representations for an entity

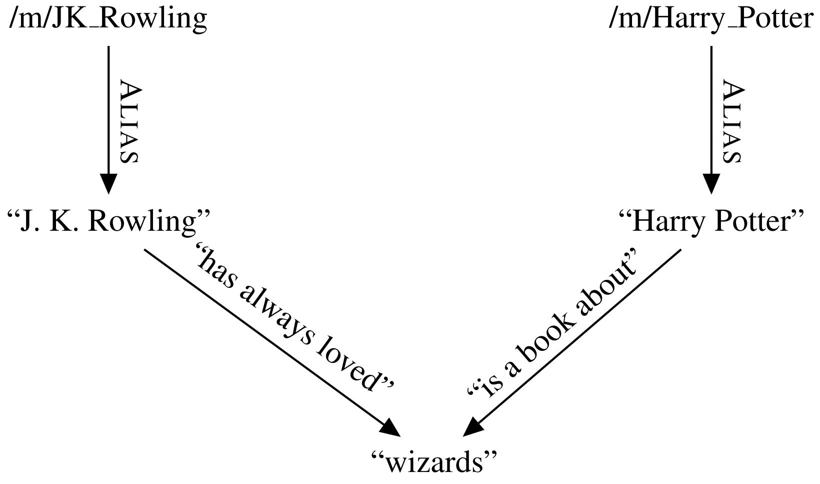


Figure 2.1: An example graph containing a noun phrase ("wizards") not linkable to a KB entity, but still potentially useful for inference.

based on its immediate neighbors, but cannot readily capture information that is separated by an ALIAS relation.

In this thesis, we will generally take this second approach when combining formal and textual knowledge bases, as our initial focus is on compositional inference techniques which can easily handle the long distance dependencies involved. By using this method, we can make use of relations between noun phrases that have no clear link to entities in the formal KB, instead of just discarding the unlinkable relations.

2.1.2 KB representations for inference

The there are several ways to represent a knowledge base. Recall that we have defined a KB as a set of triples, (e_s, r, e_t). In this section we will briefly formalize the conversion between these triples and the representations used by the inference techniques presented in Section 2.2. There are two such representations: as a graph, and as a tensor.

A KB graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$ is defined as a collection of nodes, edges, and edge types. Producing this graph from the knowledge base triples is straightforward. Each unique entity in the

KB triples corresponds to a node in the graph (i.e., $\mathcal{N} = \{e_s | (e_s, r, e_t) \in KB\} \cup \{e_t | (e_s, r, e_t) \in KB\}$). The edge types correspond to relation types in the KB ($\mathcal{R} = \{r | (e_s, r, e_t) \in KB\}$). Finally, each triple in the KB creates a single edge of type r between the node corresponding to e_s and the node corresponding to e_t.

The representation of a KB as a rank-3 tensor $\mathbf{K}$ is also straightforward. This is a binary tensor (with values that are either 0 or 1), where the first of the tensor modes corresponds to source entities in the KB, the second corresponds to target entities, and the third corresponds to relation types. The entry $\mathbf{K}_{ijk}$ is 1 if and only if (e_i, r_k, e_j) is in the set of KB triples.

2.2 Prior techniques for KB inference

We have previously defined "knowledge base inference" as the task of filling in facts that are missing from a KB. Before discussing methods for performing this task, we first give a more formal definition.

Assume there exists some "true" knowledge base $\mathcal{K}$, which, as discussed in Section 2.1, is a set of triples (e_s, r, e_t). This "true" KB is not observed; instead, we observe some incomplete (and potentially noisy) KB $\tilde{\mathcal{K}}$; that is, $\tilde{\mathcal{K}}$ does not contain all of the triples in $\mathcal{K}$ and may contain some triples not in $\mathcal{K}$. The goal of KB inference is to find the set of missing facts $\mathcal{F} = \mathcal{K} - \tilde{\mathcal{K}}$.[1]

The remainder of this section presents various approaches to finding $\mathcal{F}$. Each of these methods makes some assumptions (either explicit or implicit) about the nature of $\mathcal{K}$. The logical inference methods, for instance, assume that there is a set of rules encodable in first-order (or some other) logic that generate some or all of the facts in $\mathcal{K}$, and then try to recover those rules. Graph-based methods, including PRA, assume that there is some common substructure of the graph around pairs of nodes that are connected by the same edge type, and try to model that sub-

[1] One could also try to find the set of facts in $\tilde{\mathcal{K}}$ which are not true (i.e., $\tilde{\mathcal{K}} - \mathcal{K}$). The methods we present in this thesis could easily be used for this task also, though we do not explicitly evaluate performance on it.

structure using a set of features extracted from the graph.[2] The success of all of these methods hinges on how well the assumptions made about $\mathcal{K}$ match the actual KB that is being modeled.

2.2.1 Logical inference

Overarching this problem of KB inference is the field of statistical relational learning (SRL). SRL is the general problem of learning statistical models over relational data (including what we call knowledge bases here), and there have been a recent book (Getoor and Taskar, 2007) and survey paper (Getoor and Mihalkova, 2011) written on the subject, containing many references to tasks and methods that are related to the work in this thesis. We discuss just a few of these methods here.

An important class of SRL methods for reasoning with first order logic is based on Markov logic networks (Richardson and Domingos, 2006) (MLNs). A Markov logic network takes a set of logical rules and forms a template for a Markov random field which can be *grounded* given a knowledge base. The grounding process creates a vertex in the Markov random field for each ground atom in the KB.[3] These vertices are connected by factors corresponding to rules that apply to each ground atom. Given weights on these rules, inference over the undirected graphical model can give probabilities for each possible fact being true. MLNs are very powerful—any rule in first-order logic can be included in an MLN—but they do not scale well to very large knowledge bases, as *every ground atom* must be represented as a vertex in the grounded Markov random field. Recent work has attempted to improve the scalability of MLNs (Khot et al., 2015; Niu et al., 2011), though it remains a challenging problem for very large knowledge bases.

Another subclass of SRL methods is that of inductive logic programming (ILP) (Muggleton and De Raedt, 1994). Inductive logic programming seeks to learn relational knowledge (such as logic programs) from examples. A well-known implementation of ILP is the First Order Inductive Learner (FOIL), which performs an expensive search over a set of known facts to

[2] We will see in Section 3.5 that these two assumptions are isomorphic in certain conditions.

[3] A ground atom is essentially a possible fact, such as CITYINSTATE(PITTSBURGH, PENNSYLVANIA).

find Horn clause rules (Quinlan, 1990). FOIL can be effective at finding general rules on small datasets, but, like MLNs, it has a difficult time scaling to knowledge bases the size of those in use today.

A recent attempt to overcome the scalability issues with grounding a very large knowledge base is the ProPPR system (Wang et al., 2013). Given a set of logical inference rules, this method, Programming with Personalized PageRank, uses random walks through a "proof space" to locally ground a query in time that is independent of the size of the KB. This allows ProPPR to run successfully on much larger KBs than prior systems.

While these methods for logical inference are all very expressive, they each encounter problems when trying to infer missing facts in a very large knowledge base. Either the methods cannot tractably perform inference because of the large size of the KB (MLNs and FOIL), or they require the set of inference rules to be specified by hand prior to running inference (MLNs and the original ProPPR[4]). In the following section we discuss the path ranking algorithm, a technique that can both find inference rules and perform tractable inference over a very large KB. It accomplishes this by restricting the set of allowable inference rules to Horn clauses whose antecedents, or preconditions, correspond to chains in the graph.

2.2.2 The Path Ranking Algorithm (PRA)

The path ranking algorithm (PRA) is a technique for performing link prediction in a graph, introduced by Lao and Cohen (2010b). As noted above, the assumption made by PRA is that there is common substructure around node pairs that share the same edge label. PRA tries to model this substructure for each relation in the KB by extracting features of this substructure that correspond to paths between the node pair. Given a set of node pairs from a graph as training examples, PRA performs random walks over the graph to find sequences of edge types (or *paths*)

[4] More recent work with ProPPR has included extending it to also perform *structure learning*, allowing ProPPR to learn certain kinds of inference rules automatically (Wang et al., 2014a).

that can predict new instances of the relation represented in the training examples. The algorithm then uses logistic regression to rank these paths as features over node pairs in a prediction model. For example, in predicting new instances of the relation CityInCountry, PRA might assign a high weight to the edge type sequence -CityInState-StateInCountry-.

PRA has a strong connection to logical inference, as each of the features used by PRA can be viewed as a particular kind of Horn clause. Instead of using some kind of generative model to combine the probabilities of several Horn clauses together, however, PRA uses training data and a discriminative model to combine the evidence from multiple Horn clauses. This connection to logical inference is explored more thoroughly in Section 3.5.

Comparing PRA to other modern techniques for KB inference, we call it a *compositional* inference technique because it allows for arbitrary compositions of the relations in the KB in order to predict new relation instances. While this gives PRA a lot of representational power, it can also lead to significant feature sparsity: if there are many relation types in the KB that have similar meaning, PRA will treat each of them as independent symbols, without any means of sharing information about their similarity. These independent symbols then get composed into paths, compounding the sparsity problem.

The key novelty of PRA was in its method for constructing a feature matrix over node pairs in a graph. While Lao only ever used this feature matrix with a logistic regression model for performing link prediction, the feature matrix computed by PRA has wider applicability, as we show in Chapter 4. In this section, then, we focus on the process PRA uses to compute its feature matrix.

Consider a graph $\mathcal{G}$ with nodes $\mathcal{N}$, edges $\mathcal{E}$, and edge labels $\mathcal{R}$, and a set of node pairs (s_j, t_j) $\in \mathcal{D}$ that are instances of a target relation r. PRA will generate a feature vector for each (s_j, t_j) pair, where each feature is some sequence of edge labels -e_1-e_2-. . .-e_l-. If the edge sequence corresponding to the feature exists between the source and target nodes in the graph, the value of that feature in the feature vector will be non-zero.

Because the feature space considered by PRA is so large,[5] and because computing the feature values is so computationally intensive, the first step PRA must perform is *feature selection*, which is done using an initial search of the graph. In this step of PRA, we find path types π that are likely to be useful in predicting new instances of the target relation r. These path types are found by performing a search over the graph $\mathcal{G}$ starting at the source and target nodes in $\mathcal{D}$ and recording which paths connect some source node with its target.[6] Note that this is a *two-sided*, *unconstrained* search: the search from sources and targets can be joined on intermediate nodes to get a larger set of potential paths that connect the source and target nodes. Once connectivity statistics have been computed in this way, k path types are selected as features. Lao et al. (2011) use measures of the precision and recall of each feature in this selection, while in the work in this thesis (see Chapter 3), we simply pick those most frequently seen.

Once a set of path features has been selected, the second step of PRA is to compute values for each cell in the feature matrix. Rows in this matrix correspond to node pairs, and the columns correspond to the path types found in the first step. The value computed by PRA is the probability of arriving at the target node of a node pair, given that a random walk began at the source node and was constrained to follow the path type: $\mathrm{p}(t|s,\pi)$. There are several ways of computing this probability. The most straightforward method is to use a path-constrained breadth-first search to exhaustively enumerate all possible targets given a source node and a path type, count how frequently each target is seen, and normalize the distribution. This calculates the desired probability exactly, but at the cost of doing a breadth-first search (with complexity proportional to the average per-edge-label out-degree to the power of the path length) per source node per path type.

[5] The feature space consists of the set of all possible edge label sequences, with cardinality $\sum_{i=1}^{l} |\mathcal{R}|^i$, assuming a bound l on the maximum path length.

[6] The initial work on PRA exhaustively enumerated all possible paths up to a given length, but when moving to larger graphs corresponding to the NELL and Freebase KBs, this more constrained approach was used instead (Lao et al., 2011).

There are three methods that can potentially reduce the computational complexity of this probability calculation. The first is to use random walks to approximate the probability via rejection sampling: for each path type and source node, a number of random walks (or some kind of particle filtering) are performed, attempting to follow the edge sequence corresponding to the path type. If a node is reached where it is no longer possible to follow the path type, the random walk is restarted. This does not reduce the time necessary to get an arbitrarily good approximation, but it does allow us to decrease computation time, even getting a fixed complexity[7], at the cost of accepting some error in our probability estimates. Second, Lao (2012) showed that when the target node of a query is known, the exponent can be cut in half by using a two-sided BFS. In this method, some careful bookkeeping is done with dynamic programming such that the probability can be computed correctly when the two-sided search meets at an intermediate node. Lao's dynamic programming technique is only applicable when the target node is known, however, and only cuts the exponent in half—this is still quite computationally intensive. Lastly, we could replace the BFS with a multiplication of adjacency matrices, which performs the same computation. The efficiency gain comes from the fact that we can just do the multiplication once per path type, instead of once per path type per source node. However, to correctly compute the probabilities for a (source, target) pair, we need to exclude from the graph the edge connecting that training instance. This means that the matrix computed for each path type *should* be different for each training instance, and so we either lose our efficiency gain or we accept incorrect probability estimates. When presenting experimental results for PRA in this thesis, we use the rejection sampling technique to compute PRA's probabilities.

As mentioned above, once the feature matrix has been computed in the second step of PRA, one can use any kind of classifier desired to learn a model and make predictions on test data. All work that we are aware of that makes use of PRA has used logistic regression as the classifier here, and in this thesis we do the same.

[7] Indeed, Lao (2012) explicitly used a *time* cutoff when approximating these probabilities, so the time complexity is fixed and bounded.

Algorithm and example

```
input : A graph $\mathcal{G}$, a set of training data $D = (s_j, t_j)$
output: A set of path type features $\pi_i$, with logistic regression weight $w_i$
1. $\boldsymbol{\pi} \leftarrow$ FindPaths($D$, $\mathcal{G}$);
2. $\mathbf{M} \leftarrow$ ComputeFeatureMatrix($D$, $\boldsymbol{\pi}$, $\mathcal{G}$);
3. $\mathbf{w} \leftarrow$ TrainLogisticRegression($\mathbf{M}$);
4. return $\boldsymbol{\pi}, \mathbf{w}$;
```

Algorithm 1: The Path Ranking Algorithm (PRA)

PRA is summarized in Algorithm 1. `FindPaths` is the first step described above, where random walks (or a BFS) are performed to find edge sequences π that connect source nodes to target nodes in D. `ComputeFeatureMatrix` is the second step described above, where the probabilities $\mathrm{p}(t|s, \pi)$ are computed using one of the methods we discussed. The resultant matrix $\mathbf{M}$ has one row for each pair in D, with columns corresponding to the selected path types π. `TrainLogisticRegression` performs standard (and typically regularized) logistic regression on the data matrix $\mathbf{M}$, returning a weight for each path type π. The resultant probability model for node pair j is $\mathrm{p}(x_j) = \sigma(\mathbf{w}^T \mathbf{M}_j)$, where x_j denotes that node pair j is a positive instance of the target relation. To make predictions with a trained PRA model, `ComputeFeatureMatrix` is run on a new test data set, given the path types already computed at training time, and the probability $\mathrm{p}(x_j)$ is evaluated for each test example.

For an example of how this algorithm works, consider Figure 2.2. We will walk through the process of learning a model to predict new instances of the relation CITYINCOUNTRY (the green edge in the graph). PRA will first take all of the training instances for this relation (e.g., (Pittsburgh, U.S.), (Harrisburg, U.S.), (Tokyo, Japan), etc.) and run the `FindPaths` step. This will find edge sequences that go from the source node to the target node of the training instance, without using the training edge. For the training instance (Pittsburgh, U.S.), example paths

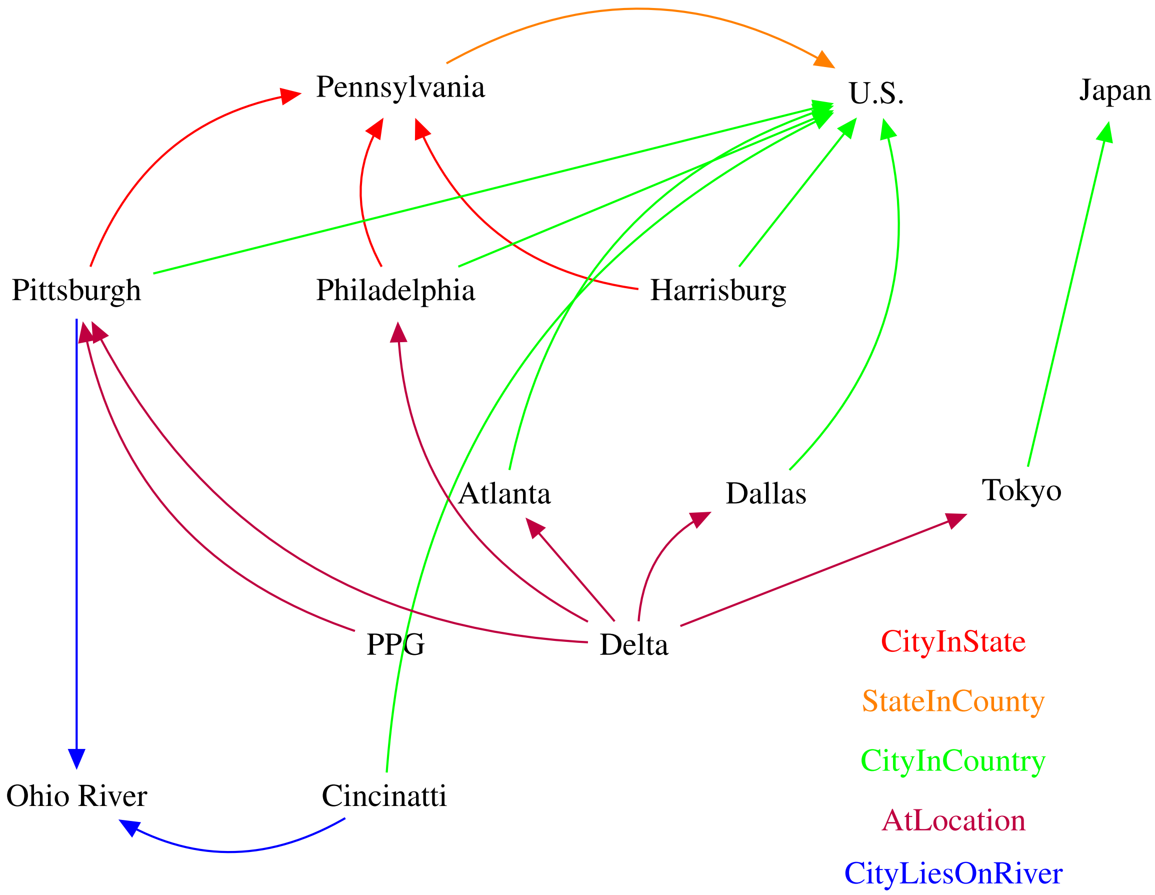


Figure 2.2: An example graph for use in explaining the path ranking algorithm. The colored edges each represent a different edge label, as shown in the lower right.

that PRA might find in this graph include -CityInState-StateInCountry-, -CityInState-CityInState^{-1}-CityInCountry-, -CityLiesOnRiver-CityLiesOnRiver^{-1}-CityInCountry-, and -AtLocation^{-1}-AtLocation-CityInCountry-. After iterating over all of the training data, a global count of path types is found, and `FindPaths` selects some number of these path types to use as features in the model.

The next step is `ComputeFeatureMatrix`, using these selected path types as features. In this step, for each training instance and selected path type, PRA computes a probability. That probability is the probability that a random walk starting at the source node of the training instance and constrained to follow the path type arrives at the target node: $\mathrm{p}(t|s,\pi)$. For the training instance (Pittsburgh, U.S.) and the path type, -CityInState-StateInCountry-, this probability is 1.0. The probability is also 1.0 for the path types -CityInState-CityInState^{-1}-CityInCountry- and -CityLiesOnRiver-CityLiesOnRiver^{-1}-CityInCountry-. For the path type -AtLocation^{-1}-AtLocation-CityInCountry-, however, the random walk will occasionally arrive in Japan, and so the probability will be somewhat lower, perhaps 0.75. `ComputeFeatureMatrix` computes these probabilities for every training instance and feature type, resulting in a feature matrix that can be used with standard logistic regression, or any other classification model.

Negative Evidence

Our discussion of PRA thus far has not included how the training data, positive or negative, is obtained. Because a knowledge base is a set of triples, it is generally trivial to obtain positive training data for any relation in the KB—simply find all triples containing that relation. Obtaining negative training examples is less trivial.

The method used originally in PRA to find negative evidence, and the method used in most of this thesis, is to keep track of all target nodes reached during the second step of the algorithm (the `ComputeFeatureMatrix` step in Algorithm 1). Consider a particular training instance

(s, t). During `ComputeFeatureMatrix`, random walks will be performed from s following each path type selected in the first step of PRA. Those random walks will end at a set of nodes T. For each node t' in T not equal to t, we create a negative example (s, t'), with features computed in the same manner as the features for the positive example (s, t).

It is most often the case that there are many more negative examples found in this manner than there are positive examples, and so generally some kind of subsampling is performed so the training data is more balanced. However, there are also many fewer non-zero features per negative example than there are per positive example, and so some care should be taken when doing the subsampling. Simply sampling random instances to use as negative examples may give a skewed distribution over the feature space; it is often more effective to use all of the negative examples but to decrease their relative weight in the objective function during training.

2.2.3 Vector space methods

A separate family of techniques for knowledge base completion learns some kind of vector space representation for the entities and relations in the KB, and uses those representations to predict which facts are missing. Many of these methods view the KB as a tensor and perform an explicit factorization and reconstruction of the tensor to fill in missing values, while other methods are more similar to neural networks in how the vector space representations are obtained and used. We describe a few of these methods in more detail below.

The first tensor factorization method we look at is called RESCAL (Nickel et al., 2011). This method models each slice of the KB tensor $\mathbf{K}$ as a product of an entity matrix, a relation matrix, and the entity matrix transposed:

$$\mathbf{C}_r \approx \mathbf{V}\mathbf{R}_r\mathbf{V}^T$$

where $\mathbf{C}_r$ is an $E \times E$ slice of the KB tensor corresponding to relation r, or the *connectivity matrix* for relation r. $\mathbf{V}$ is an $E \times d$ matrix, with one vector of length d for each entity, and $\mathbf{R}_r$

is a $d \times d$ matrix holding the latent representation for relation r. The parameters in $\mathbf{V}$ and each $\mathbf{R}_r$ are optimized to reconstruct the original tensor:

$$\min_{\mathbf{V},\mathbf{R}} \frac{1}{2} \left(\sum_r ||\mathbf{C}_r - \mathbf{V}\mathbf{R}_r\mathbf{V}^T||_F^2 \right) + \lambda_{\mathbf{V}} ||\mathbf{V}||_F^2 + \lambda_{\mathbf{R}} \sum_r ||\mathbf{R}_r||_F^2$$

A second factorization approach, called TransE (Bordes et al., 2013), models relations as translations from a source vector to a target vector: $\mathbf{v}_s + \mathbf{r} \approx \mathbf{v}_t$. The vectors are learned by minimizing a ranking hinge loss:

$$\min_{\mathbf{v},\mathbf{r}} \sum_{(s,r,t) \in KB} \max \left(0, \gamma + ||\mathbf{v}_s + \mathbf{r}_r - \mathbf{v}_t||_2^2 - ||\mathbf{v}'_s + \mathbf{r}_r - \mathbf{v}'_t||_2^2 \right)$$

where γ is a margin parameter, and $\mathbf{v}'_s$ and $\mathbf{v}'_t$ are some scrambled version of the triple in the KB, with either the source or target entity changed. The optimization thus seeks to put correct triples closer together in vector space than incorrect triples.

We mention these two methods specifically because they are particularly popular in the recent literature, and because they obtain representations that can be composed in a similar manner to how PRA composes relations.[8] There is, however, a whole host of other methods along these lines. A nice line of work that predates RESCAL and TransE is one of performing a latent clustering of the KB tensor, then using this clustering to fill in facts that are missing from KB. This research includes the infinite relational model of Kemp et al. (2006), the Bayesian clustered tensor factorization of Sutskever et al. (2009), and the max-fargin latent feature models of Zhu (2012).

[8] For a more detailed discussion of composing the representations obtained with these methods, see the work by Guu et al. (2015). Prior to the publication of Guu's work, we performed some preliminary experiments attempting to compose RESCAL representations as a replacement for the adjacency matrix computation in PRA. Our experiments were not successful, and Guu's nice paper gives some evidence for why—we did not train RESCAL to be compositional, and so composing the relation matrices did not work very well.

2.2.4 Other techniques

In Section 2.2.3, we mentioned two factorization techniques for performing knowledge base inference. There are many other similar approaches, including the structured embeddings of Bordes et al. (2011), the neural tensor networks of Socher et al. (2013), the matrix factorization over universal schemas of Riedel et al. (2013), and several recent improvements on RESCAL and TransE, the methods we discussed previously (Chang et al., 2014; García-Durán et al., 2014; Wang et al., 2014c).

If a knowledge base is defined to include natural language text, as we have done here, systems that perform inference directly over text (without some formal language as an intermediary) are also related to this work. One popular line of research on textual inference is natural logic (Angeli and Manning, 2014; MacCartney and Manning, 2007) and related systems (Schoenmackers et al., 2008).

2.3 Other related work

There is a large body of work that is related to what is presented in this thesis, from work in logic and knowledge representation, to relation extraction and other applications of natural language processing, so much that we cannot possibly hope to cover all of it. We have already discussed prior methods for knowledge base completion; in the remainder of this chapter we discuss methods for relation extraction, a task that is very closely linked to knowledge base completion, and other related work.

2.3.1 Relation extraction

Relation extraction is the task of translating some relationship between entities expressed in text into the formal language of a given knowledge base. For example, the sentence “Barack Obama is the president of the United States” might be translated into PRESIDENTOF(“Barack Obama”, "United States"). This is related to our task of knowledge base inference in that both tasks attempt to predict new instances of KB relations; relation extraction predicts these from individual sentences, while KB inference predicts these from the KB itself. The canonical examples of early relation extraction research are the Message Understanding (MUC) Conferences (Grishman and Sundheim, 1996), and the Automatic Content Extraction (ACE) program (Doddington et al., 2004).

The lines between relation extraction and KB completion become very blurred, however, when the KB itself contains textual predicates. Many (and perhaps all) approaches to KB inference can make a prediction given a single textual predicate as an example, and thus can be used directly on the relation extraction task (the matrix factorization technique of Riedel et al. (2013) is one such example, and PRA could also be constrained to make predictions in this setting). It is typically more difficult for techniques focused on relation extraction to make direct use of information from a formal knowledge base, but Riedel's work gives one example of how this can be done. Because of the similarity between these two tasks, we believe it is worthwhile to highlight a few recent approaches to relation extraction.

Initial work from the MUC and ACE programs trained models for relation extraction from sentences that were manually labeled as positive or negative examples for a given KB predicate. This kind of annotation is very costly, particularly as the number of predicates being learned increases. Recent work has thus made use of the rise of large knowledge bases to *distantly supervise* the training of relation extraction models (Hoffmann et al., 2011; Mintz et al., 2009; Surdeanu et al., 2012). These techniques find sentences in a text corpus that contain mentions of pairs of entities from the KB, then assume that (at least some of) those sentences express the relationship seen between the entities in the KB. This has allowed relation extraction research to scale to much larger domains than was previously possible, as much less manual supervision is needed to learn prediction models.

There has also been some very recent work on explicitly including a knowledge base in relation extraction models. We have already mentioned the work of Riedel et al. (2013); in addition to this, there has been work by Weston et al. (2013) and Wang et al. (2014b). These methods for relation extraction are very close to the methods we discuss for KB inference, as there is very little that separates the two tasks when the KB is used at prediction time for relation extraction.

We also mention in passing the line of work related to *open information extraction*, which seeks to extract relations from text, but not translate them into the language of a formal KB (Fader et al., 2011; Mausam et al., 2012; Yates et al., 2007). Because this work does not seek to predict new instances of a particular predicate, only find the arguments to predicative words and phrases in text, it is much more akin to parsing or semantic role labeling than to the relation extraction task we have defined here. However, we bring these techniques up as related to this work because we use the output of these systems as textual triples when we construct our knowledge bases.

2.3.2 Analysis of graphs

The path ranking algorithm, as well as the improvements to it that we present in this thesis, performs link prediction in a graph using a particular kind of random walk over that graph. There are many other methods that also use random walks over graphs or other graph properties to aid in various prediction tasks. One of the most salient pieces of work for analyzing graphs in recent time is the PageRank algorithm, the basis Google's web search engine (Page et al., 1999), and many variations on it, including Personalized PageRank (Haveliwala, 2002; Jeh and Widom, 2003; Tong et al., 2006), which we make use of in this thesis. Graphs are so general a representation that have been studied for such a long time that we cannot hope to give any kind of complete summary of all of the work that extracts properties from graphs for use in some kind of prediction model. We mention here only a few pieces of work that are particularly relevant to this thesis.

A significant portion of the work using graph representations for machine learning has focused on information retrieval tasks, where one wants to rank the nodes of a graph in response to some particular query (the PageRank algorithm is a prime example of this). Indeed, while we use PRA for link prediction or knowledge base completion in this thesis, the work leading up to PRA's development was focused on information retrieval. There were many systems for retrieving entities from a relational database in response to a user's query (Agrawal et al., 2002; Bhalotia et al., 2002; Chakrabarti, 2007). These methods used various graph properties as part of a keyword search to rank the entities in the database, though they did not use random walks and did not use a trained machine learning model. There was also work by Einat Minkov and coauthors that constructed similarity methods over graphs using random walks, and used them in reranking for name disambiguation and threading in email conversations (Minkov et al., 2006), as well as for extracting named entity coordinate terms (Minkov and Cohen, 2008). The combination of these ideas led directly to Lao and Cohen's original proposal of the path ranking algorithm (Lao and Cohen, 2010a), a method for using path-constrained random walks as features in a discriminatively trained retrieval model. Much of Lao's work focused on developing techniques for efficiently computing the random walk probabilities used by PRA (Lao, 2012), while in this thesis we focus on the feature matrix computed by PRA and explore how we can obtain better features, as well as using the algorithm in novel applications.

We also have a short discussion later in this thesis (Section 3.5) about the connection between the graph properties extracted by PRA and logic statements. This correspondence has been known for a long time, and has been exploited in practical applications at least as early as 1992, with Richards' and Mooney's work on relational pathfinding (Richards and Mooney, 1992).

Chapter 3

Reasoning with Knowledge Graphs

3.1 Introduction

In this chapter we present methods for reasoning with knowledge bases. Our primary interest with this reasoning is to perform inference: assigning probabilities to facts that are missing in the knowledge base, with the goal of adding facts that the reasoning system predicts with high confidence. We call this task *knowledge base completion* or *knowledge base inference*, and we gave a formal definition of this task in Section 2.2. For convenience, we repeat that definition here:

Assume there exists some "true" knowledge base $\mathcal{K}$, which, as discussed in Section 2.1, is a set of triples (e_s, r, e_t). This "true" KB is not observed; instead, we observe some incomplete (and potentially noisy) KB $\tilde{\mathcal{K}}$; that is, $\tilde{\mathcal{K}}$ does not contain all of the triples in $\mathcal{K}$ and may contain some triples not in $\mathcal{K}$. The goal of KB inference is to find the set of missing facts $\mathcal{F} = \mathcal{K} - \tilde{\mathcal{K}}$.

The formal definition, then, is given $\tilde{\mathcal{K}}$, find $\mathcal{F}$. There are several variations on this general theme. For instance, we could additionally be given a corpus of text related to $\mathcal{K}$ which should help us find $\mathcal{F}$. We could also simply be asked, given $\mathcal{K}$ and some triple (e_s, r, e_t) not in $\mathcal{K}$, with what probability (e_s, r, e_t) is in $\mathcal{F}$. This original formulation requires some kind of search procedure to generate candidate triples, while the second requires only a model that scores candidates. We will use a few of these variations on the knowledge base completion task to evaluate the methods presented in this chapter.

As discussed in Section 2.2, there are many methods for performing this task. The work presented in this chapter expands on prior graph-based reasoning techniques, exemplified by the path ranking algorithm (PRA) (Lao and Cohen, 2010b). Much of our work builds directly on PRA; however, when describing the techniques in this chapter, we do not explain the details of PRA again. We refer the reader back to Section 2.2.2 for a more in-depth description of the algorithm.

We present three successive improvements to PRA in this chapter. The first two of these improvements are focused on the case when the knowledge base graph contains both formal KB predicates and open-domain predicates extracted from text. Open-domain predicates can make inference challenging, as there are frequently many ways to express the same relationship in natural language, and they are treated as separate symbols by PRA. In Section 3.2, we show that clustering the open domain predicates can provide a significant improvement in inference performance. The clustering is performed over a vector space learned from a factorization of the KB tensor, and so this technique is able to combine the graph-based approach of PRA with the tensor factorization approaches to KB inference presented in Section 2.2.3.

The method in Section 3.2 still leaves us with inference over symbolic predicates, and the nature of symbolic predicates means that the algorithm is prone to sparsity, where only some predicates are seen in the training data and others are seen in test data. Section 3.3 shows how to make more complete use of the vector space representations for each relation. Instead of using the relation vectors to modify the graph and running the original PRA algorithm, the method introduced in Section 3.3 modifies the random walks in PRA to use the relation vectors directly. This leads to significantly improved performance.

The last method we present moves from incorporating vector space information into PRA to replacing PRA with a simpler, more efficient, and more expressive reasoning algorithm. Section 3.4 describes this method, which we call *subgraph feature extraction* (SFE). Instead of the complicated and computationally expensive random walk procedure defined by PRA, SFE does a simple breadth-first search to characterize a subgraph around a pair of nodes in a graph, then runs a number of feature extractors over that graph to obtain a feature matrix that can be used in machine learning models. This method is an order of magnitude faster than PRA, and it gives substantially better performance on knowledge base completion on the KBs we experimented with.

Finally, Section 3.5 presents some discussion of the connection between logical inference techniques and the techniques we have presented in this chapter, including the original path ranking algorithm. One way of viewing PRA and SFE is as extracting Horn clause rules from graphs to use in inferring missing facts in a KB, and in this section we give a formal description of that connection.

We note here also that PRA and SFE need not only be used as a kind of logical inference for predicting missing facts in a knowledge base. More generally, these are techniques for producing feature matrices over node pairs in a graph. A feature matrix can be used in many kinds of models, and the following two chapters explore other applications of these feature matrices.

3.2 Clustering relations in PRA

The content in this section is largely derived from a paper titled “Improving Learning and Inference in a Large Knowledge-base using Latent Syntactic Cues” (Gardner et al., 2013).

3.2.1 Introduction

The path ranking algorithm, as it was originally used, found inference rules that used the predicates in a formal knowledge base to predict new instances of the KB relations. PRA’s main application was to filling in missing facts in an incomplete KB (Dong et al., 2014; Lao et al.,

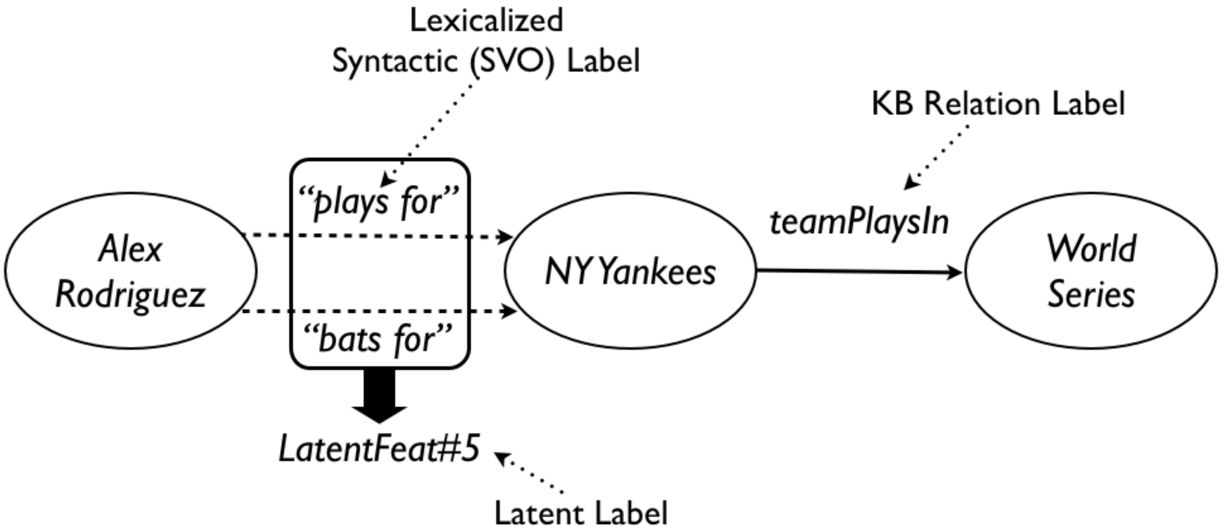


Figure 3.1: Example demonstrating how lexicalized syntactic edges can improve connectivity in the KB, enabling PRA to discover relationships between ALEX RODRIGUEZ and WORLD SERIES. Edges with latent labels can improve inference performance by reducing data sparsity. See Section 3.2.1 for details.

2011). For example, if the KB contained the facts (TIGER WOODS, PARTICIPATESIN, PGA TOUR)) and (GOLF, SPORTOFTOURNAMENT, PGA TOUR), then by putting these two facts together, we could potentially infer that (TIGER WOODS, PLAYSSPORT, GOLF). As discussed in Section 2.2.2, the path ranking algorithm (PRA) (Lao and Cohen, 2010b) performs such inference by automatically learning semantic inference rules over the KB (Lao et al., 2011). PRA uses features based off of sequences of edge types, e.g., -PLAYSSPORT-SPORTOFTOURNAMENT-, to predict missing facts in the KB. One problem with this approach, however, is that if two entities are not closely connected by intermediate facts in the formal knowledge base, PRA cannot make good predictions about their relationship. Given that the KB is (by assumption) incomplete, this can significantly hamper attempts at inferring missing facts.

To mitigate this issue, PRA was extended by Lao et al. (2012) to perform inference over a KB augmented with dependency parsed sentences. While this opens up the possibility of learning inference rules that combine information from both the formal KB and from syntactic structures, the set of syntactic edge labels used are just the *unlexicalized* dependency role labels (e.g., NOBJ, DOBJ, etc., without the corresponding words). The rules themselves thus lack much of the information contained in the text, including, crucially, the *actual words* used in the syntactic relationship. To overcome this limitation, in this section we augment the KB graph by adding edges with more expressive *lexicalized* syntactic labels (where the labels are words instead of dependencies). These additional edges, e.g., (ALEX RODRIGUEZ, "plays for", NY YANKEES), are mined by extracting 600 million Subject-Verb-Object (SVO) triples[1] from a large corpus of 500m dependency parsed documents, which would have been prohibitively expensive to add directly as done by Lao et al. (2012).

Adding edges to our inference graph with labels corresponding to natural language text brings its own problems, however. Recall that PRA uses sequences of edge labels as features in its model, and the number of possible features is the number of edge labels to the power of the sequence length. When we add natural language text as possible edge labels, we increase the number of relations by a factor 100 or more, increasing the number of potential features by 10^8. Thankfully, most of these features are never actually seen, but these additional edge labels do significantly increase the sparsity that PRA has to handle.

In order to overcome the explosion of path features and data sparsity, we derive edge labels by performing a clustering or latent embedding of the lexicalized edges. This amounts to replacing the natural language edge label with some latent cluster id, adding less than 100 edge types instead of hundreds of thousands. Through extensive experiments on real world datasets, we demonstrate effectiveness of the proposed approach.

Motivating Example

In Figure 3.1, the KB graph (only solid edges) is disconnected, thereby making it impossible for PRA to discover any relationship between ALEX RODRIGUEZ and WORLD SERIES. However,

[1] http://rtw.ml.cmu.edu/resources/svo/

addition of the two edges with SVO-based lexicalized syntactic edges (e.g., (ALEX RODRIGUEZ, "plays for", NY YANKEES)) restores this inference possibility. For example, PRA might use the edge sequence -"plays for"-TEAMPLAYSIN- as evidence for predicting the relation instance (ALEX RODRIGUEZ, ATHLETEWONCHAMPIONSHIP, WORLD SERIES). Unfortunately, such naïve addition of lexicalized edges may result in significant data sparsity, which can be overcome by mapping lexicalized edge labels to some latent embedding (e.g., (ALEX RODRIGUEZ, LATENTFEAT#5, NY YANKEES) and running PRA over this augmented graph. Using latent embeddings, PRA could then use the following edge sequence as a feature in its prediction models: -LATENTFEAT#5-TEAMPLAYSIN-. We find this strategy to be very effective as described in Section 3.2.3.

The remainder of this section is organized as follows. In Section 3.2.2 we describe how we incorporate lexical edges into the graph used by PRA, followed by our method for using a latent clustering of these lexical edges to decrease the sparsity in the graph. Section 3.2.3 then shows the results of experiments with this new technique on knowledge base inference over 15 NELL relations. Section 3.2.4 gives some discussion of the results, and Section 3.2.5 concludes.

3.2.2 Methods

$\mathrm{PRA}_{\mathrm{syntactic}}$

Recall that in Section 2.1.2 we defined a graph corresponding to a knowledge base as $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$, a collection of nodes, edges, and edge types. Let this graph $\mathcal{G}$ correspond to the nodes and edges from the formal knowledge base.

In this section, we extend the knowledge graph $\mathcal{G}$ with an augmented graph $\mathcal{G}^{'} = (\mathcal{N}, \mathcal{E}^{'}, \mathcal{R}^{'})$, where $\mathcal{E} \subset \mathcal{E}^{'}$ and $\mathcal{R} \subset \mathcal{R}^{'}$, with the set of vertices unchanged.

In order to get the edges in $\mathcal{E}^{'} - \mathcal{E}$, we first collect a set of Subject-Verb-Object (SVO) triples $D = \{(s, v, o, c)\}$ from a large dependency parsed text corpus, with $c \in \mathbb{R}_{+}$ denoting the frequency of this triple in the corpus. The additional edge set is then defined as $\mathcal{E}_{\mathrm{syntactic}} =$

$\mathcal{E}^{'} - \mathcal{E} = \{(s, v, o) \mid \exists(s, v, o, c) \in D,\ s, o \in \mathcal{N}\}$. We define $S = \{v \mid \exists(s, v, o) \in \mathcal{E}_{\text{syntactic}}\}$ and set $\mathcal{R}^{'} = \mathcal{R} \cup S$. In other words, for each pair of directly connected nodes in the KB graph $\mathcal{G}$, we add an additional edge between those two nodes for each verb which takes the NPs represented by two nodes[2] as subjects and objects (or vice versa) as observed in a text corpus. In Figure 3.1, (ALEX RODRIGUEZ, "plays for", NY YANKEES) is an example of such an edge.

PRA is then applied over this augmented graph $\mathcal{G}^{'}$. We shall refer to this version of PRA as $\text{PRA}_{\text{syntactic}}$. For the experiments in this section, we collected $|D| = 600$ million SVO triples[3] from the entire ClueWeb corpus (Callan et al., 2009), parsed using the Malt parser (Nivre et al., 2007) by the Hazy project (Kumar et al., 2013). We then filtered the 600 million triples to keep only those that shared a subject or an object with a relation instance in the training or test data.

$\text{PRA}_{\text{latent}}$

In this section we construct $\mathcal{G}^{''} = (\mathcal{N}, \mathcal{E}'', \mathcal{R}^{''})$, another syntactic-information-induced extension of the knowledge graph $\mathcal{G}$, but instead of using the surface forms of verbs in S (see previous section) as edge types, we derive those edges types $\mathcal{R}^{''}$ based on latent embeddings of those verbs. We note that $\mathcal{E} \subset \mathcal{E}^{''}$, and $\mathcal{R} \subset \mathcal{R}^{''}$.

In order to learn the latent or low dimensional embeddings of the verbs in S, we first define $Q_S = \{(s, o) \mid \exists(s, v, o, c) \in D,\ v \in S\}$, the set of subject-object tuples in D which are connected by at least one verb in S. We now construct a matrix $X_{|S|\times|Q_S|}$ whose entry $X_{v,q} = c$, where $v \in S, q = (s, o) \in Q_S$, and $(s, v, o, c) \in D$. After row normalizing and centering matrix X,[4] we apply PCA on this matrix. To obtain the principal components, we use a truncated singular value decomposition of X, giving $X = U\Sigma V$. We then form a matrix of verb representations

[2] Note here that this requires some kind of entity linking, as the nodes in $\mathcal{G}$ correspond to formal KB entities. We use a simple heuristic entity linking, using the CANREFERTO relation in NELL to obtain a set of candidate entities for each NP, then add edges between the cross product of the NPs corresponding to the subject and object.

[3] This data and other resources from the paper are publicly available at http://rtw.ml.cmu.edu/emnlp2013_pra/.

[4] For each row in X, we normalize it so it sums to 1. Then, for each column, we subtract the mean of the column to center the matrix.

	Precision	Recall	F1
PRA	0.800	0.331	0.468
$\mathrm{PRA}_{\text{syntactic}}$	0.804	0.271	0.405
$\mathrm{PRA}_{\text{latent}_c}$	**0.885**	0.334	0.485
$\mathrm{PRA}_{\text{latent}_s}$	0.868	**0.424**	**0.570**

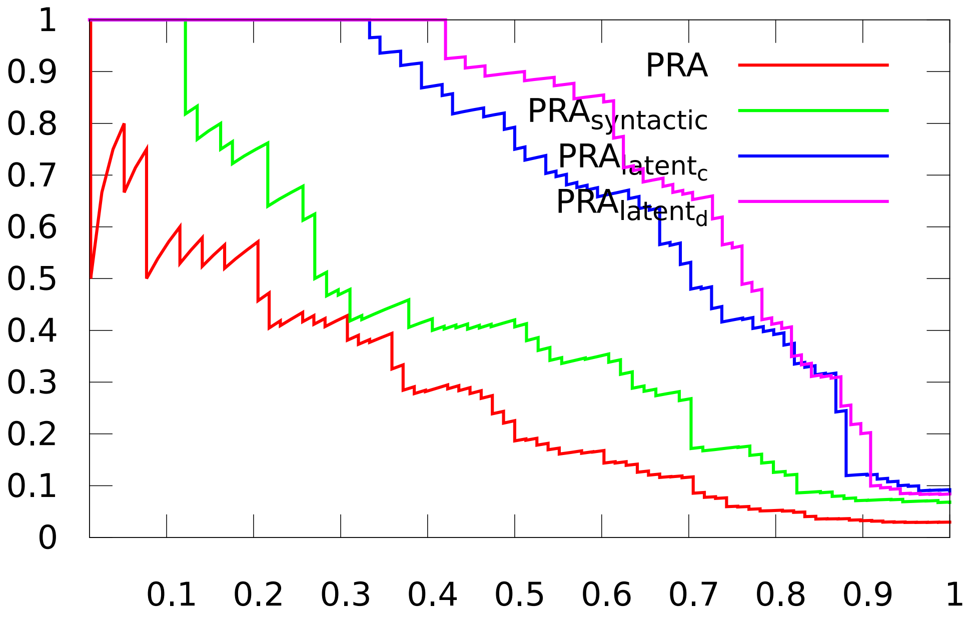
Table 3.1: Comparison of performance of different variants of PRA micro averaged across 15 NELL relations. We find that use of latent edge labels, in particular the proposed approach $\mathrm{PRA}_{\text{latent}_s}$, significantly outperforms other approaches. This is our main result. (See Section 3.2.3)

$A_{|S| \times d} = U\Sigma^{\frac{1}{2}}$, with $d << |Q_S|$ being the size of the truncated SVD. We use two strategies to derive mappings for verbs from this matrix A.

- $\mathrm{PRA}_{\text{latent}_c}$: The verb is mapped to concatenation of the (sorted) k columns with the highest absolute value in the row in A that corresponds to the verb. The sign is also kept as part of this representation. For example, with $k = 2$, the vector [.3, -.5, .1] would be mapped to -L2:L1.
- $\mathrm{PRA}_{\text{latent}_s}$: The verb is mapped to set of top-k most positive and negative columns in the row in A that corresponds to the verb. For example, with $k = 2$ the vector [.3, -.5, .1] would be mapped to the two edge labels -L2 and L1.

3.2.3 Experiments

We compared the various methods using 15 NELL relations. For each relation, we split NELL's known relation instances into 90% training and 10% testing. For each method, we then selected 750 path features and trained the model, as described in Section 2.2.2, using GraphChi (Kyrola et al., 2012) to perform the random walk graph computations. To evaluate the model, we took all source nodes in the testing data and used the model to predict target nodes. We report the



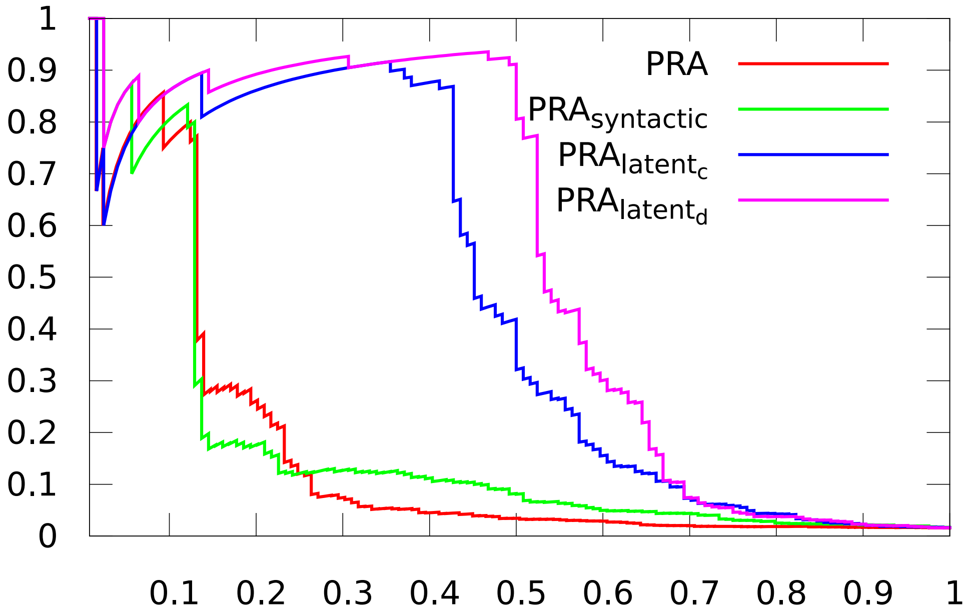


Figure 3.2: Precision (y axis) - Recall (x axis) plots for the relations CITYLIESONRIVER (top) and ATHLETEPLAYSFORTEAM (bottom). $\mathrm{PRA}_{\mathrm{latent}_s}$ (rightmost plot), the proposed approach which exploits latent edge labels, outperforms other alternatives.

precision and recall (on the set of known target nodes) of the set of predictions for each model that are above a certain confidence threshold. Because we used strong regularization, we picked for our threshold a model score of 0.405, corresponding to 60% probability of the relation instance being true; values higher than this left many relations without any predictions. Table 3.1 contains the results.

As can be seen in the table, $\mathrm{PRA}_{\mathrm{syntactic}}$ on average performs slightly worse than PRA. We believe this decrease is primarily due to noise in the training and test data for these relations in NELL. That is, for a few of the NELL relations, there is a mismatch between the facts that are true according to the formal KB and the facts that are true according to the SVO extractions. This mismatch interacts poorly with the sparsity of the SVO extractions to decrease performance on many of the relations. When we use a set of relations with less noise in Section 3.3, we see the opposite result—SVO extractions do improve performance, though the increase is quite small.

However, when using latent factorization methods to reduce the sparsity of the syntactic features, we do see a significant improvement in performance even with the noise in these NELL relations. $\mathrm{PRA}_{\mathrm{latent}_c}$ has a 45% reduction in precision errors vs. PRA while maintaining the same recall, and $\mathrm{PRA}_{\mathrm{latent}_s}$ reduces precision errors by 35% while improving recall by 27%. Section 3.2.4 contains some qualitative analysis of how sparsity is reduced with the latent methods. As a piece quantitative analysis, there were 908 possible path types found in the feature selection step with PRA on the relation CITYLIESONRIVER (of which we then selected 750). For $\mathrm{PRA}_{\mathrm{syntactic}}$, there were 73,820, while $\mathrm{PRA}_{\mathrm{latent}_c}$ had 47,554 and $\mathrm{PRA}_{\mathrm{latent}_s}$ had 58,414.

Table 3.2 shows F1 scores for each model on each relation, and Figure 3.2 shows representative Precision-Recall plots for two NELL relations. In both cases, we find that $\mathrm{PRA}_{\mathrm{latent}_s}$ significantly outperforms other baselines.

	PRA	$\text{PRA}_{\text{syntactic}}$	$\text{PRA}_{\text{latent}_c}$	$\text{PRA}_{\text{latent}_s}$
animalIsTypeOfAnimal	0.52	0.50	0.47	**0.53**
athletePlaysForTeam	0.22	0.21	0.56	**0.64**
athletePlaysInLeague	**0.81**	0.75	0.73	0.74
cityLiesOnRiver	0.05	0	0.07	**0.31**
cityLocatedInCountry	0.15	0.20	0.45	**0.55**
companyCeo	0.29	0.18	0.25	**0.35**
countryHasCompanyOffice	0	0	0	0
drugHasSideEffect	**0.96**	0.95	0.94	0.94
headquarteredIn	0.31	0.11	0.41	**0.64**
locationLocatedWithinLocation	0.40	0.38	0.38	**0.41**
publicationJournalist	0.10	0.06	0.10	**0.16**
roomCanContainFurniture	0.72	0.70	0.71	**0.73**
stadiumLocatedInCity	0.53	0	0.13	**0.67**
teamPlaysAgainstTeam	**0.47**	0.24	0.26	0.21
writerWroteBook	0.59	0.62	0.73	**0.80**

Table 3.2: F1 performance of different variants of PRA for all 15 relations tested.

3.2.4 Discussion

While examining the model weights for each of the methods, we saw a few occasions where surface relations and NELL relations combined to form interpretable path types. For example, in ATHLETEPLAYSFORTEAM, some highly weighted features took the form of -ATHLETEPLAYSSPORT-"(sport) played by (team)"-. A high weight on this feature would bias the prediction towards teams that are known to play the same sport as the athlete.

For PRA, the top features for the best performing relations are path types that contain a single edge which is a supertype or subtype of the relation being predicted. For instance, for the relation ATHLETEPLAYSFORTEAM (shown in Figure 3.2), the highest-weighted features in PRA are -ATHLETELEDSPORTSTEAM- (more specific than ATHLETEPLAYSFORTEAM) and -PERSONBELONGSTOORGANIZATION- (more general than ATHLETEPLAYSFORTEAM). For the same relation, $\mathrm{PRA}_{\mathrm{syntactic}}$ has features like -"scored for"-, -"signed"-, -"have"-, and -"led"-. When using a latent embedding of these verb phrases, "signed", "have", and "led" all have the same representation in the latent space, and so it seems clear that $\mathrm{PRA}_{\mathrm{latent}}$ gains a lot by reducing the sparsity inherent in using surface verb forms.

For CITYLIESONRIVER, where PRA does not perform as well, there is no NELL relation that is an immediate supertype or subtype, and so PRA does not have as much evidence to use. It finds features that, e.g., are analogous to the statement "cities in the same state probably lie on the same river". Adding lexical labels gives the model edges to use like "lies on", "runs through", "flows through", "starts in" and "reaches", and features using these edges give a significant boost in performance to $\mathrm{PRA}_{\mathrm{syntactic}}$. Once again, almost all of those verb phrases share the same latent embedding, and so $\mathrm{PRA}_{\mathrm{latent}}$ gains another significant boost in performance by combining them into a single feature.

3.2.5 Conclusion

In this section, we introduced the use of latent lexical edge labels for PRA-based inference over knowledge bases. We obtained such latent edge labels by mining a large dependency parsed corpus of 500 million web documents and performing PCA on the result. We gave experimental evidence that the proposed approach significantly outperforms previous state-of-the-art baselines.

3.3 Vector space random walks

The content in this section is largely derived from a paper titled "Incorporating vector space similarity in random walk inference over knowledge bases" (Gardner et al., 2014).

3.3.1 Introduction

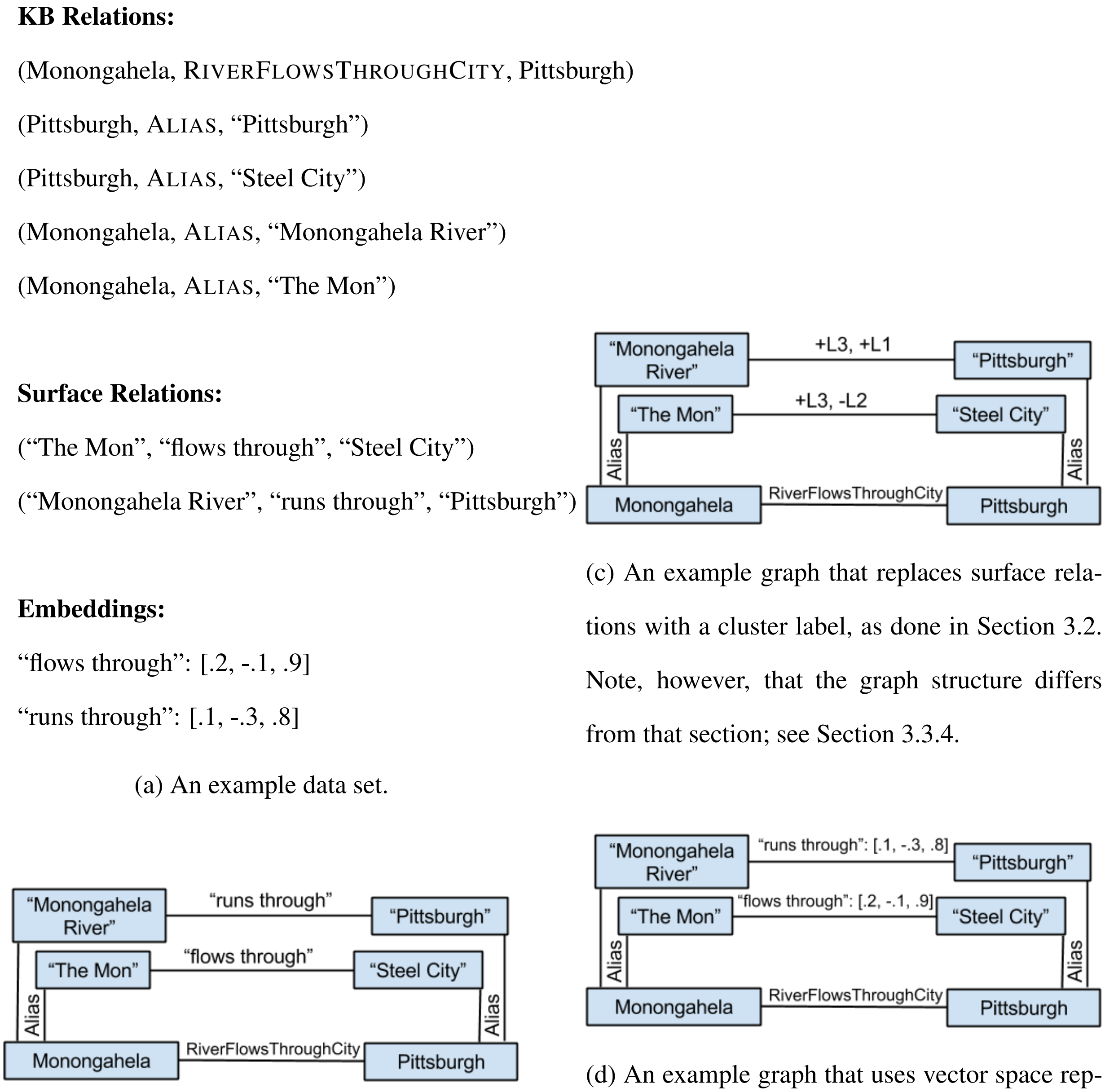
As we saw in the previous section, one major deficiency of PRA is the connectivity of the knowledge base graph—if there is no path connecting two nodes in the graph, PRA cannot predict any relation instance between them. Thus prior work has introduced the use of a text corpus to increase the connectivity of the graph used as input to PRA (Gardner et al., 2013; Lao et al., 2012). This approach is not without its own problems, however. Whereas knowledge base relations are semantically coherent and different relations have distinct meanings, this is not true of surface text. For example, "The Nile *flows through* Cairo" and "The Nile *runs through* Cairo" have very similar if not identical meaning. Adding a text corpus to the inference graph increases connectivity, but it also dramatically increases feature sparsity.

We introduce two new techniques for making better use of a text corpus for knowledge base inference. First, we describe a new way of incorporating the text corpus into the knowledge base graph that enables much more efficient processing than prior techniques, allowing us to approach problems that prior work could not feasibly solve. Second, we introduce the use of vector space similarity in random walk inference in order to reduce the sparsity of surface forms. That is, when following a sequence of edge types in a random walk on a graph, we allow the walk to follow edges that are semantically similar to the given edge types, as defined by some vector space embedding of the edge types. If a path calls for an edge of type "flows through", for example, we accept other edge types (such as "runs through") with probability proportional to the vector space similarity between the two edge types. This lets us combine notions of distributional similarity with symbolic logical inference, with the result of decreasing the sparsity of the feature space considered by PRA. We show with experiments using both the NELL and Freebase knowledge bases that this method gives significantly better performance than prior approaches to incorporating text data into random walk inference.

3.3.2 Graph Construction

Our method for knowledge base inference, described in Section 2.2.2, performs random walks over a graph to obtain features for a logistic regression classifier. Here describe how we produce the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$ from a set of knowledge base (KB) relation instances and a set of surface relation instances extracted from a corpus. Producing a graph from a knowledge base is straightforward: the set of nodes $\mathcal{N}$ is made up of the entities in the KB; the set of edge types $\mathcal{R}$ is the set of relation types in the KB, and the typed edges $\mathcal{E}$ correspond to relation instances from the KB, with one edge of type r connecting entity nodes for each (n_1, r, n_2) triple in the KB. Less straightforward is how to construct a graph from a corpus, and how to connect that graph to the KB graph. We describe our methods for each of those below.

To create a graph from a corpus, we first preprocess the corpus to obtain a collection of *surface relations*, such as those extracted by open information extraction systems like OLLIE (Mausam et al., 2012). These surface relations consist of a pair of noun phrases in the corpus, and the verb-like connection between them (either an actual verb, as done by Talukdar et al. (2012), a dependency path, as done by Riedel et al. (2013), or OpenIE relations (Mausam



KB Relations:

(Monongahela, RIVERFLOWSTHROUGHCITY, Pittsburgh)

(Pittsburgh, ALIAS, “Pittsburgh”)

(Pittsburgh, ALIAS, “Steel City”)

(Monongahela, ALIAS, “Monongahela River”)

(Monongahela, ALIAS, “The Mon”)

Surface Relations:

(“The Mon”, “flows through”, “Steel City”)

(“Monongahela River”, “runs through”, “Pittsburgh”)

Embeddings:

“flows through”: [.2, -.1, .9]

“runs through”: [.1, -.3, .8]

(a) An example data set.

(b) An example graph that combines a KB and surface relations.

(c) An example graph that replaces surface relations with a cluster label, as done in Section 3.2. Note, however, that the graph structure differs from that section; see Section 3.3.4.

(d) An example graph that uses vector space representations of surface edges, as introduced in this section.

Figure 3.3: Example graph construction as used in the experiments in this section. A graph using only KB edges is simply a subset of these graphs containing only the RIVERFLOWSTHROUGHCITY edge, and is not shown.

et al., 2012)). The verb-like connections are naturally represented as edges in the graph, as they have a similar semantics to the knowledge base relations that are already represented as edges. We thus create a graph from these triples exactly as we do from a KB, with nodes corresponding to noun phrase types and edges corresponding to surface relation triples.

So far these two subgraphs we have created are entirely disconnected, with the KB graph containing nodes representing entities, and the surface relation graph containing nodes representing noun phrases, with no edges between these noun phrases and entities. We connect these two graphs by making use of the ALIAS relation in the KB, which links entities to potential noun phrase referents. Each noun phrase in the surface relation graph is connected to those entity nodes which the noun phrase can possibly refer to according to the KB. These edges are not the output of an entity linking system, as done by Lao et al. (2012), but express instead the notion that the noun phrase *can* refer to the KB entity. The use of an entity linking system would certainly allow a stronger connection between noun phrase nodes and entity nodes, but it would require much more preprocessing and a much larger graph representation, as each mention of each noun phrase would need its own node, as opposed to letting every mention of the same noun phrase share the same node. This graph representation allows us to add tens of millions of surface relations to a graph of tens of millions of KB relations, and perform all of the processing on a single machine.

As will be discussed in more detail in Section 3.3.3, we also allow edge types to optionally have an associated vector that ideally captures something of the semantics of the edge type.

Figure 3.3 shows the graph constructions used in our experiments on a subset of KB and surface relations. Note that Figure 3.3b and Figure 3.3c are shown as rough analogues of graphs used in prior work (described in more detail in Section 3.3.4), and we use them for comparison in our experiments.

3.3.3 Method

Our modifications to PRA are confined entirely to the feature computation step described in Section 2.2.2; feature selection (finding potentially useful sequences of edge types) proceeds as normal, using the symbolic edge types. When computing feature values, however, we allow a walk to follow an edge that is *semantically similar* to the edge type in the path, as defined by Euclidean distance in the vector space.

More formally, consider a path type π. Recall that π is a sequence of edge types -e_1-e_2-...-e_l-, where l is the length of the path; we will use π_i to denote the i^{th} edge type in the sequence. To compute feature values, PRA begins at some node and follows edges of type π_i until the sequence is finished and a target node has been reached. Specifically, if a random walk is at a node n with m outgoing edge types $\{e_1, e_2, \ldots, e_m\}$, PRA selects the edge type from that set which matches π_i, then selects uniformally at random from all outgoing edges of that type. If there is no match in the set, the random walk restarts from the original start node.

We modify the selection of which edge type to follow. When a random walk is at a node n with m outgoing edge types $\{e_1, e_2, \ldots, e_m\}$, instead of selecting only the edge type that matches π_i, we allow the walk to select instead an edge that is close to π_i in vector space. For each edge type at node n, we select the edge with the following probability:

$$p(e_j|\pi_i) \propto \exp(\beta \times v(e_j) \cdot v(\pi_i)), \; \forall j, 1 \leq j \leq m$$

where $v(\cdot)$ is a function that returns the vector representation of an edge type, and β is a spikiness parameter that determines how much weight to give to the vector space similarity. As β approaches infinity, the normalized exponential approximates a delta function on the closest edge type to π_i, in $\{e_1, e_2, \ldots, e_m\}$. If π_i is in the set of outgoing edges, this algorithm converges to the original PRA.

However, if π_i is not in the set of outgoing edge types at a node and all of the edge types are very dissimilar to π_i, this algorithm (with β not close to infinity) will lead to a largely uniform distribution over edge types at that node, and no way for the random walk to restart. To recover the restart behavior of the original PRA, we introduce an additional restart parameter α, and add another value to the categorical distribution before normalization:

$$p(\text{restart}|\pi_i) \propto \exp(\beta * \alpha)$$

When this *restart* type is selected, the random walk begins again, following π_1 starting at the source node. With α set to a value greater than the maximum similarity between (non-identical) edge type vectors, and β set to infinity, this algorithm exactly replicates the original PRA.

Not all edge types have vector space representations: the ALIAS relation cannot have a meaningful vector representation, and we do not use vectors to represent KB relations, finding that doing so was not useful in practice (which makes intuitive sense to us: KB relations are already latent representations themselves). While performing random walks, if π_i has no vector representation, we fall back to the original PRA algorithm for selecting the next edge.

We note here that when working with vector spaces it is natural to try clustering the vectors to reduce the parameter space. Each path type π is a feature in our model, and if two path types differ only in one edge type, and the differing edge types have very similar vectors, the resultant feature values will be essentially identical for both path types. It seems reasonable that running a simple clustering algorithm over these path types, to reduce the number of near-duplicate features, would improve performance. We did not find this to be the case, however; all attempts we made to use clustering over these vectors gave performance indistinguishable from not using clustering. From this we conclude that the main issue hindering performance when using PRA over these kinds of graphs is one of limited connectivity, not one of too many parameters in the model. Though the feature space considered by PRA is very large, the number of attested features in a real graph is much smaller, and it is this sparsity which our vector space methods address.

3.3.4 Prior work relevant to this section

Knowledge base inference. The kind of random walk inference over knowledge bases we are dealing with here was first introduced by Lao and Cohen (2010b). This work was improved upon shortly afterward to also make use of a large corpus, by representing the corpus as a graph and connecting it to the knowledge base (Lao et al., 2012). We further showed in Section 3.2 that replacing surface relation labels with a representation of a latent embedding of the relation led to improved prediction performance. This result is intuitive: the feature space considered by PRA is exponentially large, and surface relations are sparse. The relations "[river] flows through [city]" and "[river] runs through [city]" have near identical meaning, and both should be very predictive for the knowledge base relation RIVERFLOWSTHROUGHCITY. However, if one of these relations only appears in the training data and the other only appears in the test data, neither will be useful for prediction.

We attempted to solve this issue in Section 3.2 by finding a latent symbolic representation of the surface relations (such as a clustering) and replacing the edge labels in the graph with these latent representations. This makes it more likely for surface relations seen in training data to also be seen at test time, and naturally improved performance. This representation, however, is still brittle, as it is still a symbolic representation that is prone to mismatches between training and test data. If the clustering algorithm used is too coarse, the features will not be useful, and if it is too fine, there will be more mismatches. Also, verbs that are on the boundaries of several clusters are problematic to represent in this manner. The key difference between the method discussed in Section 3.2 and what we present in this section is that here we directly use the vector representations of edge types during the random walk inference, instead of doing an ad-hoc symbolic replacement.

These two prior techniques are the most directly related work to what we present in this section, and we compare our work to theirs.

Graph construction. In addition to the incorporation of vector space similarity into the PRA algorithm, the major difference between our work and the prior approaches mentioned above is in the construction of the graph used by PRA. We contrast our method of graph construction with these prior approaches in more detail below.

Lao et al. (2012) represent every word of every sentence in the corpus as a node in the graph, with edges between the nodes representing dependency relationships between the words. They then connect this graph to the KB graph using a simple entity linking system (combined with coreference resolution). The resultant graph is enormous, such that they needed to do complex indexing on the graph and use a cluster of 500 machines to perform the PRA computations. Also, as the edges represent dependency labels, not words, with this graph representation the PRA algorithm does not have access to the verbs or other predicative words that appear in the corpus, which frequently express relations. PRA only uses *edge* types as feature components, not *node* types, and so the rich information contained in the words is lost. This graph construction also would not allow the incorporation of vector space similarity that we introduced, as dependency labels do not correspond to KB relations, and vector space representations of them would not be useful for this task.

The graph construction for the clustered PRA approach taken in Section 3.2 is very similar to the graph construction presented in Section 3.3.2, as both preprocess the corpus to obtain surface relations. However, instead of creating a graph with nodes representing noun phrases, Section 3.2 added edges from the surface relations directly to the entity nodes in the graph. Using the ALIAS relation, as this section does, Section 3.2 added an edge between *every possible* concept pair that could be represented by the noun phrases in a surface relation instance. This leads to some nonsensical edges added to the graph (e.g., encoding that Michael I. Jordan, the professor, plays for the Chicago Bulls). In addition, if the ALIAS relation has high degree (as it does for many common noun phrases in Freebase), it quickly becomes unscalable—this method of graph construction runs out of disk space when attempting to run on the Freebase experiments in Section 3.3.5. Also, in conflating entity nodes in the graph with noun phrases, they lose an important distinction that turns out to be useful for prediction, as we discuss in Section 3.3.5.[5]

3.3.5 Experiments

We perform both the feature selection step and the feature computation step of PRA using GraphChi, an efficient single-machine graph processing library (Kyrola et al., 2012). We use MALLET's implementation of logistic regression, with both L1 and L2 regularization (McCallum, 2002). To obtain negative evidence, we used a closed world assumption, treating any (source, target) pair found during the feature computation step as a negative example if it was not given as a positive example. We tuned the parameters to our methods using a coarse, manual grid search with cross validation on the training data described below. The parameters we tuned were the L1 and L2 regularization parameters, how many random walks to perform in the feature selection and computation steps of PRA, and spikiness and restart parameters for vector space walks.[6] The results presented were not very sensitive to changes in these parameters.

Data

We ran experiments on both the NELL and Freebase knowledge bases. The characteristics of these knowledge bases are shown in Table 3.3. The Freebase KB is very large; to make it slightly more manageable we filtered out relations that did not seem applicable to relation extraction, as

[5] Recent notions of "universal schema" (Riedel et al., 2013) also put KB entities and noun phrases into the same conceptual space, though they opt for using noun phrases instead of the KB entities used by Gardner et al. In general this is problematic, as it relies on some kind of entity linking system as preprocessing, and cannot handle common noun references of proper entities without losing information. Our method, and that of Lao et al., skirts this issue entirely by not trying to merge KB entities with noun phrases.

[6] The L1 weight was set to 0.005, the L2 weight was 1, the spikiness parameter was 3, and the restart parameter was 0.25 for NELL and 0.75 for Freebase. For finding paths in NELL, we performed 200 walks per training example for 3 iterations, and for Freebase we performed 250 walks for 4 iterations. For computing feature values, we did 50 walks per source entity in NELL, and 150 in Freebase.

	NELL	Freebase
Entities	1.2M	20M
Relation instances	3.4M	67M
Total relation types	520	4215
Relation types tested	10	24
Avg. instances/relation	810	200
SVO triples used	404k	28M

Table 3.3: Statistics of the data used in our experiments.

well as a few of the largest relations.[7] This still left a very large, mostly intact KB, as can be seen in the table. For our text corpus, we make use of a set of subject-verb-object triples extracted from dependency parses of ClueWeb documents (Talukdar et al., 2012). There are 670M such triples in the data set, most of which are completely irrelevant to the knowledge base relations we are trying to predict. For each KB, we filter the SVO triples, keeping only those which can possibly connect training and test instances of the relations we used in our experiments. The number of SVO triples kept for each KB is also shown in Table 3.3. We obtained vector space representations of these surface relations by running PCA on the SVO matrix.

We selected 10 NELL relations and 24 Freebase relations for testing our methods. The NELL relations were hand-selected as the relations with the largest number of known instances that had a reasonable precision (the NELL KB is automatically created, and some relations have low precision). We split the known instances of these relations into 75% training and 25% testing, giving on average about 650 training instances and 160 test instances for each relation.

The 24 Freebase relations were semi-randomly selected. We first filtered the 4215 relations

[7] We removed anything under /user, /common, /type (except for the relation /type/object/type), /base, and /freebase, as not applicable to our task. We also removed relations dealing with individual music tracks, book editions, and TV epidsodes, as they are very large, very specific, and unlikely to be useful for predicting the relations in our test set.

based on two criteria: the number of relation instances must be between 1000 and 10000, and there must be no mediator in the relation.[8] Once we selected the relations, we kept all instances of each relation that had some possible connection in the SVO data.[9] This left on average 200 instances per relation, which we again split 75%-25% into training and test sets.

We note here also that both NELL and Freebase has type restrictions on the domain and range for each of its relations, and we make use of this to filter the predictions made by all methods, only keeping those that are type-consistent with the constraints in the KB.

Methods

The methods we compare correspond to the graphs shown in Figure 3.3. The KB method uses the original PRA algorithm on just the KB relations, as presented by Lao and Cohen (2010b). KB + SVO adds surface relations to the graph (Figure 3.3b). We present this as roughly analogous to the methods introduced by Lao et al. (2012), though with some significant differences in graph representation, as described in Section 3.3.4. KB + Clustered SVO follows the methods introduced in Section 3.2, but using the graph construction introduced in this section (Figure 3.3c; the graph construction techniques in Section 3.2 would have made graphs too large to be feasible for the Freebase experiments). KB + Vector SVO is our method (Figure 3.3d).

Evaluation

As evaluation metrics, we use *mean average precision* (MAP) and *mean reciprocal rank* (MRR), following recent work evaluating relation extraction performance (West et al., 2014). Both of

[8] A mediator in Freebase is a reified relation instance meant to handle n-ary relations, for instance /film/performance. PRA in general, and our implementation of it in particular, needs some modification to be well-suited to predicting relations with mediators.

[9] We first tried randomly selecting instances from these relations, but found that the probability of selecting an instance that benefited from an SVO connection was negligible. In order to make use of the methods we present, we thus restricted ourselves to only those that had a possible SVO connection.

Method	MAP	MRR
KB	0.193	0.635
KB + SVO	0.218	0.763
KB + Clustered SVO	0.276	0.900
KB + Vector SVO	**0.301**	**0.900**

Table 3.4: Results on the NELL knowledge base. The bolded line is significantly better than all other results with $p < 0.025$.

these metrics assume that a method returns a ranked list of predictions for each relation, and that there is some set of instances that are given to be true. We can define *average precision* as the area under a precision-recall curve for one of the test relations. MAP is the mean value of the average precision for all test relations. MRR is the mean value of the *reciprocal rank* for all relations, where reciprocal rank is the inverse of the rank at which the first true prediction is found in the list of predictions. When testing statistical significance, we test the signifance of the difference between MAP values, using a paired permutation test with the average precision on each relation as paired data.

The results of these experiments are shown in Table 3.4 and Table 3.5. In Table 3.6 we show average precision for every relation tested on the NELL KB, and we show the same for Freebase in Table 3.7.

Discussion

We can see from the tables that KB + Vector SVO (the method presented in this paper) significantly outperforms prior approaches in both MAP and MRR. We believe that this is due to the reduction in feature sparsity enabled by using vector space instead of symbolic representations (as that is the only real difference between KB + Clustered SVO and KB + Vector SVO), allowing PRA to make better use of path types found in the training data. When looking at the results

Method	MAP	MRR
KB	0.278	0.614
KB + SVO	0.294	0.639
KB + Clustered SVO	0.326	0.651
KB + Vector SVO	**0.350**	**0.670**

Table 3.5: Results on the Freebase knowledge base. The bolded line is significantly better than all other results with $p < 0.0002$.

Relation	KB	KB + SVO	KB + Clustered SVO	KB + Vector SVO
ActorStarredInMovie	0.000	0.032	0.032	**0.037**
AthletePlaysForTeam	0.200	0.239	0.531	**0.589**
CityLocatedInCountry	0.126	0.169	0.255	**0.347**
JournalistWritesForPublication	0.218	0.254	0.291	**0.319**
RiverFlowsThroughCity	0.000	0.001	0.052	**0.076**
SportsTeamPositionForSport	0.217	**0.217**	0.178	0.180
StadiumLocatedInCity	0.090	0.156	0.275	**0.321**
StateHasLake	0.000	0.000	0.000	0.000
TeamPlaysInLeague	0.934	0.936	**0.947**	0.939
WriterWroteBook	0.144	0.179	0.195	**0.202**

Table 3.6: Average precision for each relation tested on the NELL KB. The best performing method on each relation is bolded.

Relation	KB	KB + SVO	KB + C-SVO	KB + V-SVO
/amusement_parks/park/rides	0.000	0.009	0.004	**0.013**
/architecture/architect/structures_designed	0.072	0.199	0.257	**0.376**
/astronomy/constellation/contains	0.004	**0.017**	0.000	0.008
/automotive/automotive_class/examples	0.003	0.001	0.002	**0.006**
/automotive/model/automotive_class	0.737	0.727	0.742	**0.768**
/aviation/airline/hubs	0.322	0.286	0.298	**0.336**
/book/literary_series/author_s	0.798	0.812	0.818	**0.830**
/computer/software_genre/software_in_genre	0.000	0.001	0.001	**0.001**
/education/field_of_study/journals_in_this_discipline	0.001	**0.003**	0.003	0.001
/film/film/rating	**0.914**	0.905	0.914	0.905
/geography/island/body_of_water	0.569	0.556	0.580	**0.602**
/geography/lake/basin_countries	0.420	0.361	0.409	**0.437**
/geography/lake/cities	0.111	0.134	**0.177**	0.175
/geography/river/cities	0.030	0.038	0.045	**0.066**
/ice_hockey/hockey_player/hockey_position	0.307	0.243	0.222	**0.364**
/location/administrative_division/country	0.989	0.988	**0.991**	0.989
/medicine/disease/symptoms	0.061	**0.078**	0.068	0.067
/medicine/drug/drug_class	**0.169**	0.164	0.135	0.157
/people/ethnicity/languages_spoken	0.134	**0.226**	0.188	0.223
/spaceflight/astronaut/missions	0.010	0.186	0.796	**0.848**
/transportation/bridge/body_of_water_spanned	0.534	0.615	0.681	**0.727**
/tv/tv_program_creator/programs_created	0.164	0.179	0.163	**0.181**
/visual_art/art_period_movement/associated_artists	0.044	0.040	**0.046**	0.037
/visual_art/visual_artist/associated_periods_or_movements	0.276	**0.295**	0.282	0.290

Table 3.7: Average precision for each relation tested on the Freebase KB. The best performing method on each relation is bolded. For space considerations, “Clustered SVO” is shortened to “C-SVO” and “Vector SVO” is shortened to “V-SVO” in the table header.

for individual relations in Table 3.6 and Table 3.7, we see that KB + Vector SVO outperforms other methods on the majority of relations, and it is a close second when it does not.

We can also see from the results that mean average precision seems a little low for all methods tested. This is because MAP is computed as the precision of *all* possible correct predictions in a ranked list, where precision is counted as 0 if the correct prediction is not included in the list. In other words, there are many relation instances in our randomly selected test set that are not inferrable from the knowledge base, and the low recall hurts the MAP metric. MRR, which judges the precision of the top prediction for each relation, gives us some confidence that the main issue here is one of recall, as MRR is reasonably high, especially on the NELL KB. As further evidence, if we compute average precision for each query node (instead of for each relation), excluding queries for which the system did not return any predictions, MAP ranges from .29 (KB) to .45 (KB + Vector SVO) on NELL (with around 30% of queries having no prediction), and from .40 (KB) to .49 (KB + Vector SVO) on Freebase, (where 21% of queries gave no prediction). Our methods thus also improve MAP when calculated in this manner, but it is not an entirely fair metric,[10] so we use standard MAP to present our main results.

One interesting phenomenon to note is a novel use of the ALIAS relation in some of the relation models. The best example of this was found with the relation /PEOPLE/ETHNICITY/LANGUAGES_SPOKEN. A high-weighted feature when adding surface relations was the edge sequence -ALIAS-ALIAS INVERSE-. This edge sequence reflects the fact that languages frequently share a name with the group of people that speaks them (e.g., Maori, French). And because PRA can generate compositional features, we also find the following edge sequence for the same relation: -/PEOPLE/ETHNICITY/INCLUDED_IN_GROUP-ALIAS-ALIAS INVERSE-. This feature captures the same notion that languages get their names from groups of people, but applies it to subgroups within an ethnicity. These features would be very difficult, perhaps impossible, to include in systems that do not distinguish between noun phrases and knowledge base entities, such as the graphs

[10] MAP is intended to include some sense of recall, but excluding queries with no predictions removes that and opens the metric to opportunistic behavior.

constructed in Section 3.2, or typical relation extraction systems, which generally only work with noun phrases after performing a heuristic entity linking.

3.3.6 Conclusion

We have offered two main contributions to the task of knowledge base inference. First, we have presented a new technique for combining knowledge base relations and surface text into a single graph representation that is much more compact than graphs used in prior work. This allowed us to apply methods introduced previously to much larger problems, running inference on a single machine over the entire Freebase KB combined with tens of millions of surface relations. Second, we have described how to incorporate vector space similarity into random walk inference over knowledge bases, reducing the feature sparsity inherent in using surface text. This allows us to combine distributional similarity with symbolic logical inference in novel and effective ways. With experiments on many relations from two separate knowledge bases, we have shown that our methods significantly outperform prior work on knowledge base inference.

The code and data used in the experiments in this section are available at
http://rtw.ml.cmu.edu/emnlp2014_vector_space_pra/.

3.4 Subgraph Feature Extraction

The content in this section is largely derived from a paper titled "Efficient and Expressive Knowledge Base Completion Using Subgraph Feature Extraction" (Gardner and Mitchell, 2015).

3.4.1 Introduction

In this section we examine more closely the particulars of the path ranking algorithm (PRA) to determine whether the algorithm can be made both more efficient and more expressive. As we have discussed in Section 2.2.2, PRA generates a feature matrix over node pairs in a graph using a two-step process. The first step finds potential path types between node pairs to use as features in a statistical model, and the second step computes random walk probabilities associated with each path type and node pair (these are the values in a feature matrix).[11] This second step is very computationally intensive, requiring time proportional to the average out-degree of the graph to the power of the path length for *each cell* in the computed feature matrix. We consider whether this computational effort is well-spent, or whether we might more profitably spend computation in other ways. We propose a new way of generating feature matrices over node pairs in a graph that aims to improve both the efficiency and the expressivity of the model relative to PRA.

Our technique, which we call *subgraph feature extraction* (SFE), is similar to only doing the first step of PRA. Given a set of node pairs in a graph, we first do a local search to characterize the graph around each node. We then run a set of feature extractors over these local subgraphs to obtain feature vectors for each node pair. In the simplest case, where the feature extractors only look for paths connecting the two nodes, the feature space is equivalent to PRA's, and this is the same as running PRA and binarizing the resultant feature vectors. However, because we do not have to compute random walk probabilities associated with each path type in the feature matrix, we can extract much more expressive features, including features which are not representable as paths in the graph at all. In addition, we can do a more exhaustive search to characterize the local graph, using a breadth-first search instead of random walks. SFE is a much simpler method than PRA for obtaining feature matrices over node pairs in a graph. Despite its simplicity, however, we show experimentally that it substantially outperforms PRA, both in terms of running time and prediction performance. SFE decreases running time over PRA by an order of magnitude, it improves mean average precision from .432 to .528 on the NELL KB, and it improves mean reciprocal rank from .850 to .933.

In the remainder of this section, we first describe PRA in more detail. We then situate our methods in the context of related work, and provide additional experimental motivation for the

[11] Note that the first step only needs to be done at training time; when performing inference the path types have already been selected, and only the second step needs to be done.

Dataset	**Method**	**MAP**
Freebase	Probabilities	.337
	Binarized	.344
NELL	Probabilities	.303
	Binarized	.319

Table 3.8: Using binary feature values instead of random walk probabilities gives statistically indistinguishable performance. The p-value on the Freebase data is .55, while it is .25 for NELL.

improvements described in this section. We then formally define SFE and the feature extractors we used, and finally we present an experimental comparison between PRA and SFE on the NELL KB. The code and data used in this section are available at http://rtw.ml.cmu.edu/emnlp2015_sfe/.

3.4.2 Motivation

We first motivate our modifications to PRA with three observations. First, it appears that binarizing the feature matrix produced by PRA, removing most of the information gained in PRA's second step, has no significant impact on prediction performance in knowledge base completion tasks. We show this on the NELL KB and the Freebase KB in Table 3.8.[12] The fact that random walk probabilities carry no additional information for this task over binary features is a surprising result, and it shows that the second step of PRA spends a lot of its computation for no discernible gain in performance.

Second, Neelakantan et al. (2015) presented experiments showing a substantial increase in performance from using a much larger set of features in a PRA-like model.[13] All of their experi-

[12] The NELL data and experimental protocol is described in Section 3.4.4. The Freebase data consists of 24 relations from the Freebase KB; we used the same data used in Section 3.3.

[13] Note that while Neelakantan et al. called the baseline they were comparing to "PRA", they only used the first step of the algorithm to produce path types, and thus did not really compare against PRA per se. It is their version of "PRA" that we formalize and expand as SFE in this work.

ments used binary features, so this is not a direct comparison of random walk probabilities versus binarized features, but it shows that increasing the feature size beyond the point that is computationally feasible with random walk probabilities seems useful. Additionally, they showed that using path bigram features, where each sequential pair of edges types in each path was added as an additional feature to the model, gave a significant increase in performance. This kind of features is not representable in the traditional formulation of PRA.

Lastly, the method used to compute the random walk probabilities—rejection sampling—makes the inclusion of more expressive features problematic. Consider the path bigrams mentioned above; one could conceivably compute a probability for a path type that only specifies that the last edge type in the path must be r, but it would be incredibly inefficient with rejection sampling, as most of the samples would end up rejected (leaving aside the additional issues of an unspecified path length). In contrast, if the features simply signify whether a particular path type exists in the graph, without any associated probability, these kinds of features are very easy to compute.

Given this motivation, our work attempts to improve both the efficiency and the expressivity of PRA by removing the second step of the algorithm. Efficiency is improved because the second step is the most computationally expensive, and expressivity is improved by allowing features that cannot be reasonably computed with rejection sampling. We show experimentally that the techniques we introduce do indeed improve performance quite substantially.

3.4.3 Method

Here we discuss how SFE constructs feature matrices over node pairs in a graph using just a single search over the graph for each node (which is comparable to only using the first step of PRA). As outlined in Section 2.2.2, the first step of PRA does a series of random walks from each source and target node (s_j, t_j) in a dataset $\mathcal{D}$. In PRA these random walks are used to find a relatively small set of potentially useful path types for which more specific random walk probabilities are then computed, at great expense. In our method, subgraph feature extraction (SFE), we stop after this first set of random walks and instead construct a binary feature matrix.

More formally, for each node n in the data (where n could be either a source node or a target node), SFE constructs a subgraph centered around that node using k random walks. Each random walk that leaves n follows some path type π and ends at some intermediate node i. We keep all of these (π, i) pairs as the characterization of the subgraph around n, and we will refer to this subgraph as G_n. To construct a feature vector for a source-target pair (s_j, t_j), SFE takes the subgraphs G_{s_j} and G_{t_j} and merges them on the intermediate nodes i. That is, if an intermediate node i is present in both G_{s_j} and G_{t_j}, SFE takes the path types π corresponding to i and combines them (reversing the path type coming from the target node t_j). If some intermediate node for the source s_j happens to be t_j, no combination of path types is necessary (and similarly if an intermediate node for the target t_j is s_j—the path only needs to be reversed in this case). This creates a feature space that is exactly the same as that constructed by PRA: sequences of edge types that connect a source node to a target node. To construct the feature vector SFE just takes all of these combined path types as binary features for (s_j, t_j). Note, however, that we need not restrict ourselves to only using the same feature space as PRA; Section 3.4.3 will examine extracting more expressive features from these subgraphs.

This method for generating a feature matrix over node pairs in a graph is much simpler and less computationally expensive than PRA, and from looking at Table 3.8 we would expect that it would perform on par with PRA with drastically reduced computation costs. Some experimentation shows that it is not that simple. Table 3.9 shows a comparison between PRA and SFE on 10 NELL relations.[14] SFE has a higher mean average precision, but the difference is not statistically significant. There is a large variance in the performance of SFE, and on some relations PRA performs better.

We examined the feature matrices computed by these methods and discovered that the reason

[14] The data and evaluation methods are described more fully in Section 3.4.4. These experiments were conducted on a different development split of the same data.

Method	**MAP**	**Ave. Features**
PRA	.3704	835
SFE	.4007	8275

Table 3.9: Comparison of PRA and SFE on 10 NELL relations. The difference shown is not statistically significant.

for the inconsistency of SFE's improvement is because its random walks are all *unconstrained*. Consider the case of a node with a very high degree, say 1000. If we only do 200 random walks from this node, we cannot possibly get a complete characterization of the graph even one step away from the node. If a particularly informative path is -CITYINSTATE-STATEINCOUNTRY-, and both the city from which a random walk starts and the intermediate state node have very high degree, the probability of actually finding this path type using unconstrained random walks is quite low. This is the benefit gained by the *path-constrained* random walks performed by PRA; PRA leverages training instances with relatively low degree and aggregation across a large number of instances to find path types that are potentially useful. Once they are found, significant computational effort goes into discovering whether each path type exists for *all* (s, t) pairs. It is this computational effort that allows the path type -CITYINSTATE-STATEINCOUNTRY- to have a non-zero value even for very highly connected nodes.

How do we mitigate this issue, so that SFE can consistently find these path types? It seems the only option without resorting to a similar two-step process to what PRA uses is to do a more exhaustive search. PRA uses random walks to improve scalability on very large graphs, particularly because the second step of the algorithm is so expensive. However, if we are only doing a single search, and the graph fits in memory, a few steps of a breadth-first search (BFS) per node is not infeasible. We can make the BFS more tractable by excluding edge labels whose fan out is too high. For example, at a type node in Freebase, there could be thousands of edges with label /TYPE/OBJECT/TYPE; if there are a large number of edges of the same label leaving

Method	MAP	Ave. Features	Time
PRA	.3704	835	44 min.
SFE-RW	.4007	8275	6 min.
SFE-BFS	.4799	237853	5 min.

Table 3.10: Comparison of PRA and SFE on 10 NELL relations. SFE-RW is not statistically better than PRA, but SFE-BFS is ($p < 0.05$).

a node, we do not include those edges in the BFS. Note that because the type node will still be counted as an intermediate node in the subgraph, we can still find paths that go through that node; we just do not continue searching if the out-degree of a particular edge label is too high.

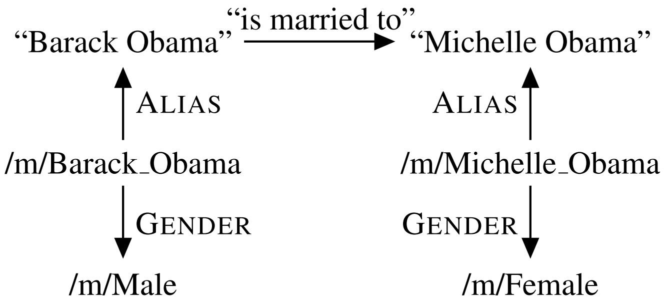
When using a BFS instead of random walks to obtain the subgraphs G_{s_j} and G_{t_j} for each node pair, we saw a dramatic increase in the number of path type features found and a substantial increase in performance.[15] These results are shown in Table 3.10; SFE-RW is our SFE implementation using random walks, and SFE-BFS uses a breadth-first search.

More expressive features

The description above shows how to recreate the feature space used by PRA using our simpler subgraph feature extraction technique. As we have mentioned, however, we need not restrict ourselves to merely recreating PRA's feature space. Eliminating random walk probabilities allows us to extract a much richer set of features from the subgraphs around each node, and here we present the feature extractors we have experimented with. Figure 3.4 contains an example graph that we will refer to when describing these features.

PRA-style features. We explained these features in Section 3.4.3, but we repeat them here for consistency, and use the example to make the feature extraction process more clear. Relying

[15] One should not read too much into the decrease in running time between SFE-RW and SFE-BFS, however, as it was mostly an implementation detail.



Subgraph for /m/Barack_Obama	
π	i
-ALIAS-	“Barack Obama”
-GENDER-	/m/Male
-ALIAS-“is married to”-	“Michelle Obama”

Subgraph for /m/Michelle_Obama	
π	i
-ALIAS-	“Michelle Obama”
-GENDER-	/m/Female
-ALIAS-“is married to”$^{-1}$-	“Barack Obama”

Figure 3.4: An example graph, with subgraphs extracted for two nodes.

on the notation introduced earlier, these features are generated by intersecting the subgraphs G_s and G_t on the intermediate nodes. That is, when the subgraphs share an intermediate node, we combine the path types found from the source and target to that node. In the example in Figure 3.4, there are two common intermediate nodes ("Barack Obama" and "Michelle Obama"), and combining the path types corresponding to those nodes gives the same path type: -ALIAS-"is married to"-ALIAS^{-1}-.

Path bigram features. In Section 3.4.2, we mentioned that Neelakantan et al. (2015) experimented with using path bigrams as features. We include those features here as well. Formally, for any path π between a source node s and a target node t, we create a feature for each relation bigram in the path type. In the example in Figure 3.4, this would result in the features "BIGRAM:@START@-ALIAS", "BIGRAM:ALIAS-is married to", "BIGRAM:is married to-ALIAS", and "BIGRAM:ALIAS-@END@".

One-sided features. We use *one-sided path* to describe a sequence of edges that starts at a source or target node in the data, but does not necessarily terminate at a corresponding target or source node, as PRA features do. Following the notation introduced in Section 3.4.3, we use as features each (π, i) pair in the subgraph characterizations G_s and G_t, along with whether the feature came from the source node or the target node. The motivation for these one-sided path types is to better model which sources and targets are good candidates for participating in a particular relation. For example, not all cities participate in the relation CITYCAPITALOFCOUNTRY, even though the domain of the relation is all cities. A city that has a large number of sports teams may be more likely to be a capital city, and these one-sided features could easily capture that kind of information.

Example one-sided features from the example in Figure 3.4 would be "SOURCE:-GENDER-:male", "TARGET:-GENDER-:female", "SOURCE:-ALIAS-:Barack Obama", and "SOURCE:-ALIAS-is married to-:Michelle Obama".

One-sided feature comparisons. We can expand on the one-sided features introduced above by allowing for comparisons of these features in certain circumstances. For example, if both the source and target nodes have an age, gender, or religion encoded in the graph, we might profitably use comparisons of these values to make better predictions.

Drawing again on the notation from Section 3.4.3, we can formalize these features as analogous to the pairwise PRA features. To get the PRA features, we intersect the intermediate nodes i from the subgraphs G_s and G_t, and combine the path types π when we find common intermediate nodes. To get these comparison features, we instead intersect the subgraphs on the path types, and combine the intermediate nodes when there are common path types. That is, if we see a common path type, such as /PEOPLE/PERSON/RELIGION, we will construct a feature representing a comparison between the intermediate node for the source and the target. If the values are the same, this information can be captured with a PRA feature, but it cannot be easily captured by PRA when the values are different.

In the example in Figure 3.4, there are two common path types: -ALIAS-, and -GENDER-. The feature generated from the path type -GENDER- would be "COMPARISON:-GENDER-:/m/Male:/m/Female".

Vector space similarity features. In Section 3.3 we introduced a modification of PRA's random walks to incorporate vector space similarity between the relations in the graph. On the data we were using, a graph that combined a formal knowledge base with textual relations extracted from text, we found that this technique gave a substantial performance improvement. The vector space random walks only affected the second step of PRA, however, and we have removed that step in SFE. While it is not as conceptually clean as the vector space random walks, we can obtain a similar effect with a simple feature transformation using the vectors for each relation. We obtain vector representations of relations through factorization of the knowledge base tensor as did Gardner et al., and replace each edge type in a PRA-style path with edges that are similar to it in the vector space. We also introduce a special "any edge" symbol, and say that all other edge types are similar to this edge type.

To reduce the combinatorial explosion of the feature space that this feature extractor creates, we only allow replacing one relation at a time with a similar relation. In the example graph in Figure 3.4, and assuming that "spouse of" is found to be similar to "is married to", some of the features extracted would be the following: "VECSIM:-ALIAS-is married to-ALIAS-", "VECSIM:-ALIAS-spouse of-ALIAS-", "VECSIM:-ALIAS-@ANY_REL@-ALIAS-", and "VECSIM:-@ANY_REL@-is married to-ALIAS-". Note that the first of those features, "VECSIM:-ALIAS-is married to-ALIAS-", is necessary even though it just duplicates the original PRA-style feature. This allows path types with different but similar relations to generate the same features.

Any-Relation features. It turns out that much of the benefit gained from Gardner et al.'s vector space similarity features came from allowing any path type that used a surface edge to match *any* other surface edge with non-zero probability.[16] To test whether the vector space similarity features give us any benefit over just replacing relations with dummy symbols, we add a feature extractor that is identical to the one above, assuming an empty vector similarity mapping. The features extracted from Figure 3.4 would thus be "ANYREL:-@ANY_REL@-is married to-ALIAS", "ANYREL:-ALIAS-@ANY_REL@-ALIAS", "ANYREL:-ALIAS-is married to-@ANY_REL@".

3.4.4 Experiments

Here we present experimental results evaluating the feature extractors we presented, and a comparison between SFE and PRA. As we showed in Section 3.4.3 that using a breadth-first search to obtain subgraphs is superior to using random walks, all of the experiments presented here use the BFS implementation of SFE.

[16] Replacing all surface edges with a single dummy relation gives performance close to vector space PRA. The vector space walks do statistically outperform this, but the extra gain is small.

Data

To evaluate SFE and the feature extractors we introduced, we learned models for 10 relations from the NELL KB. We used the same data as in Section 3.3, including both the formal KB relations and the surface relations extracted from text in the graph. We used logistic regression with elastic net (L1 and L2) regularization. We tuned the L1 and L2 parameters for each method on a random development split of the data, then generated a new split of the data to run the final tests presented here.

The evaluation metric we use is mean average precision (MAP). MAP is the mean of the average precision (AP) on each relation, where AP measures the area under a precision recall curve given a ranked list of predictions and a set of true positives. We judge statistical significance using a paired permutation test, where the average precision on each relation is used as paired data.

On Obtaining Negative Evidence

One important practical issue for most uses of PRA is the selection of negative examples for training a model. Typically a knowledge base only contains positive examples of a relation, and it is not clear a priori what the best method is for obtaining negative evidence. Prior work with PRA makes a closed world assumption, treating any (s, t) pair not seen in the knowledge base as a negative example. Negative instances are selected when performing the second step of PRA—if a random walk from a source ends at a target that is not a known correct target for that source, that source-target pair is used as a negative example. Importantly, this was also how some prior work with PRA was evaluated (including the work in both Section 3.2 and Section 3.3); only source nodes were given at test time, and the task was to both generate potential targets for the source node and score the resultant (source, target) pairs to produce a ranked list of predictions.

SFE only scores (source, target) pairs; it has no mechanism similar to PRA's that will find potential targets given a source node. We thus need a new way of finding negative examples,

Method	MAP
PRA's random walks	.359
PPR-based sampling	.363

Table 3.11: Comparing methods for obtaining negative evidence available at training time. The difference seen is not statistically significant ($p = .77$).

both at training time and at test time. We used a simple technique to find negative examples from a graph given a set of positive examples, and we used this to obtain the training and testing data used in the experiments below. Our technique takes each source and target node in the given positive examples and finds other nodes in the same category that are close in terms of personalized page rank (PPR). We then sample new (source, target) pairs from these lists of similar nodes, weighted by their PPR score (while also allowing the original source and target to be sampled). These become our negative examples, both at training and at testing time.

Because this is changing the negative evidence available to PRA at training time, we wanted to be sure we were not unfairly hindering PRA in our comparisons. If it is in fact better to let PRA find its own negative examples at training time, instead of the ones sampled based on personalized page rank, then we should let PRA get its own negative evidence. We thus ran an experiment to see under which training regime PRA performs better. We created a test set with both positive and negative examples as described in the paragraph above (i.e., not in the prior way of just giving PRA source nodes at test time), and at training time we compared two techniques: (1) letting PRA find its own negative examples through its random walks, and (2) only using the negative examples selected by PPR. As can be see in Table 3.11, the difference between the two training conditions is very small, and it is not statistically significant. We thus feel justified in giving both PRA and SFE the same training data, created through the PPR-based sampling technique described above.

Feature Types	MAP	MRR	Features
PRA-style features	.431	.806	240k
+ Comparisons	.405	.833	558k
+ One-sided	.389	.800	1,227k
+ One-sided + Comps.	.387	.817	1,544k
+ Bigrams	.483	1.00	320k
+ Vector similarity	.514	.910	3,993k
+ Bigrams + vec sim.	.490	.950	4,073k
+ Any Rel	**.528**	.933	649k

Table 3.12: SFE feature ablation study. All rows use PRA features. PRA + any rel is statistically better than all other methods except PRA + vec sim, and most of the other differences are not significant.

Results

We first examine the effect of each of the feature types introduced in Section 3.4.3. The results are shown in Table 3.12. We can see that, for this data, the comparisons and one-sided features did not improve performance (and the decreases are not statistically significant). Bigram features do appear to improve performance, though the improvement was not consistent enough across relations to achieve statistical significance. The vector similarity features do improve performance, with p-values hovering right at 0.05 when comparing against only PRA features and PRA + bigram features. The any rel features, however, do statistically improve over all other methods ($p <= 0.01$) except the PRA + vec sim result ($p = .21$).

Finally, we present a comparison between PRA, PRA with vector space random walks, and the best SFE result from the ablation study. This is shown in Table 3.13. SFE significantly outperforms PRA, both with and without the vector space random walks presented in Section 3.3.

Method	MAP	MRR
PRA	.362	.717
Vector space PRA	.432	.850
SFE (PRA + any rel features)	**.528**	.933

Table 3.13: Results of final comparison between SFE and PRA, with and without vector space similarity features. SFE is statistically better than both PRA methods ($p < 0.005$).

Discussion

When using only PRA-style features with SFE, the highest weighted features were almost always those of the form -ALIAS-[some textual relation]-ALIAS^{-1}-. For example, for the relation WRITERWROTEBOOK, the textual relations used in this feature might be "wrote", "describes in", "writes in", and "expresses in". These are the same feature types that PRA itself finds to have the highest weight, also, though SFE finds many more of them than PRA does, as PRA has to do aggressive feature selection. For this particular dataset, where the graph consists of edges from a formal KB mixed with edges from extracted textual relations, these kinds of features are by far the most useful, and most of the improvements seen by the additional feature types we used with SFE come from more compactly encoding these features.

For example, the path bigram features can encode the fact that there exists a path from the source to the target that begins or ends with an ALIAS edge. This captures in just two features all path types of the form -ALIAS-[some textual relation]-ALIAS^{-1}-, and those two bigram features are almost always the highest weighted features in models where they are used.

However, the bigram features do not capture those path types exactly. The Any-Rel features were designed in part specifically for this path type, and they capture it exactly with a single feature. For all 10 relations, the feature "ANYREL:-ALIAS-@ANY_REL@-ALIAS^{-1}" is the highest weighted feature. This is because, for the relations we experimented with, knowing that *some* relationship is expressed in text between a particular pair of KB entities is a very strong indication of a single KB relation. There are only so many possible relationships between cities and countries, for instance, and so the knowledge that a city and a country occur together in the same sentence is very strong evidence that the city is located in the country. These features are much less informative between entity types where more than one relation is frequently discussed, such as between people. For example, consider that is it frequent in text to say that two people are married, that they compete with each other, that they have some professional relationship, or a host of other semantic relationships. That mere fact that two people were mentioned in the same sentence is not strong evidence in favor of a single semantic relationship, as it is for cities and countries.

While the bigram and any-rel features capture succintly whether textual relations are present between two entities, the one-sided features are more useful for determining whether an entity fits into the domain or range of a particular relation. We saw a few features that did this, capturing fine-grained entity types. Most of the features, however, tended towards memorizing (and thus overfitting) the training data, as these features contained the names of the training entities. We believe this overfitting to be the main reason these features did not improve performance, along with the fact that the relations we tested do not need much domain or range modeling (as opposed to, e.g., SPOUSEOF or CITYCAPITALOFCOUNTRY).

3.4.5 Conclusion

We have explored several practical issues that arise when using the path ranking algorithm for knowledge base completion. An analysis of several of these issues led us to propose a simpler algorithm, which we called subgraph feature extraction, which characterizes the subgraph around node pairs and extracts features from that subgraph. SFE is both significantly faster and performs better than PRA on this task. We showed experimentally that we can reduce running time by an order of magnitude, while at the same time improving mean average precision from .432 to .528 and mean reciprocal rank from .850 to .933. This thus constitutes the best published results for knowledge base completion on NELL data. The code and data used in the experiments in this section are available at http://rtw.ml.cmu.edu/emnlp2015_sfe/.

3.5 Connection to Logical Inference

Some of the content in this section is derived in part from a paper titled "Combining Vector Space Embeddings with Symbolic Logical Inference over Open-Domain Text" (Gardner et al., 2015).

3.5.1 Introduction

In this section we discuss in more detail the relationship between techniques for logical inference and the reasoning methods we have presented in this chapter. Section 3.5.2 makes explicit the informal statements we have made previously that there is a "connection" between the path ranking algorithm and Horn clause learning. Section 3.5.3 then explores how this connection changes with the modifications to PRA made in Section 3.2 and Section 3.3. Finally, Section 3.5.4 gives a similar analysis for subgraph feature extraction, which was introduced in Section 3.4.

Before entering that discussion, however, we pause to formally define the notion of a *Horn clause*. If A is an *atom* in a formal logic (e.g., *True*, PLAYSFOR(MICHAEL_JORDAN, CHICAGO_BULLS), etc.), then a Horn clause is a statement with the following form:

$$P_1 \wedge P_2 \wedge \cdots \wedge P_n \implies Q$$

where all P_i and Q are nonnegated atoms.

Horn clauses are useful because there exists a polynomial-time procedure for inference with Horn clauses, and thus they are one of the most computationally tractable ways of using formal logic for reasoning. Prolog, a logical programming language, uses Horn clauses as the basis of its computation.

3.5.2 PRA as logical inference

Many systems exist for finding weighted Horn clause rules from a set of relations (Quinlan, 1990; Schoenmackers et al., 2010). Given a set of inference rules, a set of relations, and a query, a common technique for performing logical inference is to construct a Markov logic network and use belief propagation or Monte Carlo techniques to answer the query (e.g., the Holmes system of Schoenmackers et al. (2008)). As the number of inference rules grows, however, performing inference in this manner can be quite expensive.

PRA, by contrast, both finds potential Horn clause rules and performs inference over them with simple random walks over a graph representation of the knowledge base. PRA leverages the fact that known instances of a particular relation can be treated as training examples for learning a discriminatively trained model. The inference is impoverished in two ways, however: (1) the random walks can only find Horn clause rules whose antecedents correspond to chains of edges in the graph, so the set of possible rules is restricted; and (2) the logistic regression model is not recursive; that is, all intermediate facts necessary for a proof must already be present in the knowledge base for PRA to make a correct inference.[17]

We can formalize the connection between logical inference and PRA as follows. For each relation R in the knowledge base, PRA finds a set of N Horn clause rules, each of which can be written as:

$$R(X,Y) \impliedby \left(\bigwedge_{i=0}^{k} P_i(Z_i, Z_{i+1}) \right) \wedge Z_0 = X \wedge Z_{k+1} = Y$$

where k is the number of literals in the antecedent of the Horn clause rule.

For a particular query $R(X,Y)$, we denote the value of each inference rule n with $\pi_n(X,Y)$, where $\pi_n(X,Y)$ is 1 if all of the $P_i(Z_i, Z_{i+1})$ facts in the rule are present in the knowledge base,

[17] This second issue can be mitigated by running PRA repeatedly, adding high-confidence predictions to the knowledge base after each run, as is currently done in the NELL system (Carlson et al., 2010). Additionally, ProPPR (Wang et al., 2013) uses similar techniques to PRA that are also recursive.

and 0 otherwise.[18] Instead of creating a joint (generative) model over all facts in the knowledge base and assigning a probability to each Horn clause rule, PRA creates a discriminative model for each relation and assigns a weight to each Horn clause rule using logistic regression. Thus the potentially intractable problem of combining probabilities from independently created Horn clause rules is avoided, and PRA assigns a probability to $R(X,Y)$ as follows:

$$\Pr(R(X,Y)) = \sigma\left(\sum_n w_n \pi_n(X,Y)\right)$$

where σ is the sigmoid function and w_n is the logistic regression weight for Horn clause rule n.

3.5.3 Logical inference with vector space semantics

While the connection between the original PRA algorithm and logical inference is relatively straightforward and already known, our extensions of PRA to incorporate vector space semantics make the connection less obvious. Here we make that connection explicit, showing how our changes to the algorithm in our recent work map to changes in the formulas for logical inference above. We believe this mapping is potentially instructive, showing how other logical inference methods, such as Markov logic networks, could incorporate similar modifications to make use of vector space semantics.

First we take the simpler case of clustered surface forms (Section 3.2). A clustering defines a function, $C(P_i, P_j)$, which returns true if P_i and P_j share the same cluster. With this function, we can rewrite the formula for each Horn clause inference rule as follows:

$$R(X,Y) \Longleftarrow \bigwedge_{i=0}^{k}\left(\bigvee_{S_i\ s.t.\ C(P_i,S_i)} S_i(Z_i, Z_{i+1})\right) \wedge Z_0 = X \wedge Z_{k+1} = Y$$

[18] In practice, PRA only samples from these chains in the graph, so $\pi_n(X,Y)$ may be 0 when the facts are actually present. Additionally, previous work in PRA has used random walk probabilities for $\pi_n(X,Y)$; our experimentation has shown that binarizing these values makes little difference and may actually give better performance, so we use binary values here to simplify the discussion.

The rest of the PRA algorithm remains the same. Thus this clustering technique simply adds disjunctions to every Horn clause rule[19] containing a predicate in a cluster of size larger than 1, where the clusters are constructed using some distributed representation of the predicates. This will help in instances when sparsity of training data leads to some predicates in a Horn clause rule only being seen at test time, and it will hurt when the addition of a disjunction clause to the rule decreases the rule's specificity (and thus its inferential utility). In our experiments, we found this technique to improve performance when surface forms were clustered, but not when knowledge base relations were clustered. This makes intuitive sense, as knowledge base relations already have a well-defined semantics in the context of the KB (and so adding a disjunction will only dilute the rule), while textual relations do not. It is possible that in a graph that combines multiple KBs (e.g., NELL, Freebase, YAGO, ConceptNet, etc.), using vector space representations for KB relations (and disjunctions generated from them) could improve performance.

The vector space random walk technique presented in Section 3.3 is slightly more complicated to formulate in logical notation, but still reduces essentially to the introduction of disjunctions in the Horn clause rules. This technique modifies the random walks in PRA to follow an edge in the graph at node Z_i with probability proportional to the vector space similarity between the edge label and the Horn clause predicate P_i. Thus instead of a clustering function $C(P_i, P_j)$, we define a similarity function $\mathrm{Sim}(P_i, P_j)$, which assigns a score (typically) between -1 and 1 to any relation pair in the knowledge base.[20] With this function, our disjunction now includes *all* relations in the knowledge base for every predicate, but the probability of sampling a 1 for $\pi_n(X, Y)$ depends on the similarity function. Thus

$$R(X,Y) \Longleftarrow \bigwedge_{i=0}^{k} \left(\bigvee_{S \in \mathrm{KB}} S(Z_i, Z_{i+1}) \right) \wedge Z_0 = X \wedge Z_{k+1} = Y$$

[19] Note that this makes the rule no longer a Horn clause, as Horn clauses must be conjunctions of individual atoms.

[20] If either P_i or P_j does not have a vector space representation, the function returns $-\infty$.

and

$$\Pr(\pi_n(X,Y)) \propto \prod_{i=0}^{k} \left(\sum_{S\ s.t.\ S(Z_i,Z_{i+1})} \exp\left(\mathrm{Sim}(P_i,S)\right) \right)$$

In conclusion, our use of vector space semantics to augment logical inference translates essentially to the introduction of disjunctions that are dependent in some way on similarity in the vector space (either through an explicit clustering, or through a modification to the sampling probability in PRA). Despite their simplicity, we showed these techniques to give significant improvements in performance when reasoning with surface forms (see Section 3.2 and Section 3.3). This suggests that other methods of performing logical inference could also benefit from the targeted addition of disjunctions in inference rules, where these disjunctions are determined by some function over the vector space representations of relations. Not all methods can tractably handle this addition of disjunctions, however, as PRA can.

3.5.4 SFE as logical inference

There are two main differences between PRA and SFE in their comparison to logic systems: (1) instead of the sampling probabilities and feature weights used by PRA, SFE uses a deterministic search and binary feature values;[21] and (2) the set of Horn clauses representable by SFE is much less restricted than that used by PRA.

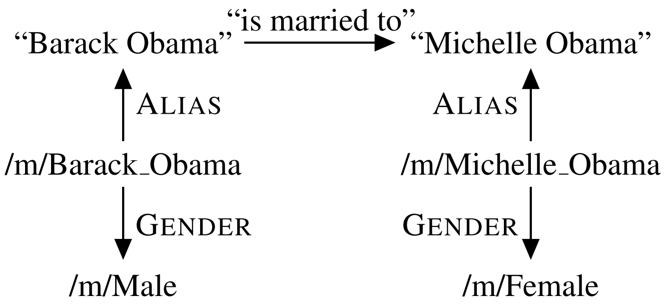
Following the notation in Section 3.5.2, then, we still have the set of Horn clauses used by PRA:

$$R(X,Y) \impliedby \left(\bigwedge_{i=0}^{k} P_i(Z_i,Z_{i+1}) \right) \wedge Z_0 = X \wedge Z_{k+1} = Y$$

with the final probability for $R(X,Y)$ still of the form

$$\Pr(R(X,Y)) = \sigma\left(\sum_n w_n \pi_n(X,Y) \right)$$

[21] The discussion in Section 3.5.2 also used binary feature values with PRA for simplicity, though the original algorithm would need a slightly more complicated analysis.



Subgraph for /m/Barack_Obama	
π	i
-ALIAS-	“Barack Obama”
-GENDER-	/m/Male
-ALIAS-“is married to”-	“Michelle Obama”

Subgraph for /m/Michelle_Obama	
π	i
-ALIAS-	“Michelle Obama”
-GENDER-	/m/Female
-ALIAS-“is married to”$^{-1}$-	“Barack Obama”

Figure 3.5: An example graph, with subgraphs extracted for two nodes.

SFE, however, add many more logical rules to the set $\{\pi_n\}$, many of which go beyond Horn clauses. For each of the feature types discussed in Section 3.4.3, we give the form of the Horn clause associated with that feature type. For easier reference during this discussion, we reproduce Figure 3.4 here as Figure 3.5.

One-sided features. The one-sided features defined by SFE consist of a beginning node (either X or Y), a path type (here a set of predicates P_i), and an ending node, which we shall denote I. Example one-sided features from the example in Figure 3.5 would be “SOURCE:-GENDER-:male”, “TARGET:-GENDER-:female”, “SOURCE:-ALIAS-:Barack Obama”, and “SOURCE:-

ALIAS-is married to-:Michelle Obama".

The Horn clause corresponding to one of these features takes a form very similar to a PRA-style feature:[22]

$$R(X,Y) \Longleftarrow \left(\bigwedge_{i=0}^{k} P_i(Z_i, Z_{i+1}) \right) \land Z_0 = X \land Z_{k+1} = I$$

The only difference between these Horn clauses and the PRA Horn clauses are the constraints on Z_0 and Z_{k+1}: in a PRA Horn clause, $Z_0 = X$ and $Z_{k+1} = Y$; here, Z_0 is either X or Y, depending on whether the feature is a source feature or a target feature, and $Z_{k+1} = I$.

One-sided feature comparisons. These features are simply the conjunction of two one-sided features with identical path types P_i. An example feature from Figure 3.5 would be "COMPARISON:-GENDER-:/m/Male:/m/Female". Using I_X and I_Y to denote the ending nodes corresponding to the source and target, respectively, we can write the Horn clause rule as follows:

$$R(X,Y) \Longleftarrow \left(\bigwedge_{i=0}^{k} P_i(Z_i, Z_{i+1}) \land P_i(S_i, S_{i+1}) \right) \land Z_0 = X \land Z_{k+1} = I_X \land S_0 = Y \land S_{k+1} = I_Y$$

Vector space similarity features. As SFE defines vector space similarity features, these are functionally equivalent to the clustered PRA Horn clauses discussed in Section 3.5.3, with the introduction of disjunctions to replace some predicates in a PRA-style Horn clause.

Any-Relation features. These feature types are also straightforward, as they are just extensions of the disjunctions mentioned above to include all possible predicates.

Path bigram features. Recall that these features take a sequence of edge types between a source and target node and extract bigrams from this sequence. Examples of these features in Figure 3.5 are "BIGRAM:@START@-ALIAS", "BIGRAM:ALIAS-is married to", "BIGRAM:is married to-ALIAS", and "BIGRAM:ALIAS-@END@".

Because these features assume the existence of a path connecting X and Y, they are more complicated to describe as a logical statement. They are not Horn clauses, but instead are state-

[22] This example corresponds to a SOURCE feature; a TARGET feature would have $Z_0 = Y$ instead of $Z_0 = X$.

ments in second-order logic. The underlying structure of the statement will look similar to a PRA Horn clause, but with free predicate variables and some additional conditions. Let j be the starting index of the bigram, ranging from -1 to k, and P_1 and P_2 the two predicates specified by the bigram feature. We first describe the case where $0 \leq j \leq k-1$. These features correspond to a second-order statement of the following form:

$$R(X,Y) \impliedby \left(\bigwedge_{i=0}^{k} Q_i(Z_i, Z_{i+1}) \right) \wedge Z_0 = X \wedge Z_{k+1} = Y \wedge Q_j = P_1 \wedge Q_{j+1} = P_2$$

where the Q_i are free variables ranging over predicates. In the case that $j = -1$, the condition $Q_j = P_1$ is dropped, as P_1 corresponds to an arbitrary start symbol, and similarly the condition $Q_{j+1} = P_2$ is dropped when $j = k$.

We can see from this discussion that SFE gives a much broader range of possible logic statement features than those allowed by PRA. While the features presented here are still a rather limited set of all possible statements, many more could be added by simply defining new feature extractors. The only conditions are that the logical statements must only use predicates and nodes that are in the local region of X and Y, and that the resultant feature space not be intractably large (i.e., there are an exponential number of possible statements that could be extracted from a subgraph, and SFE cannot handle all of them as features in any model that runs on a real machine).

3.6 Conclusion

This chapter has discussed reasoning with knowledge graphs, focusing on the task of knowledge base completion. We have shown two methods that successfully combine information from factorization models of KB completion with graph-based methods, reducing the feature sparsity in the path ranking algorithm. We also developed a third method that improves significantly over PRA in both the efficiency and the expressivity of its reasoning capabilities. Taken together, these methods represent a very substantial improvement over the previous state-of-the-art—an over 50% relative increase in mean average precision over standard PRA (see Table 3.13).

We additionally gave a formal description of the connection between the methods used in this chapter and logical reasoning systems—these methods all perform a kind of discriminative logical inference, combining information from many Horn clauses in a logistic regression model. The space of logic that these systems can capture is fundamentally limited, however. While SFE can handle a much larger set of Horn clauses than PRA, it still cannot do recursive reasoning, and there are many first-order structures that would be difficult or impossible to discover and evaluate with SFE. In exchange for this impoverished reasoning capability, SFE is able to perform very efficient inference and scale to very large knowledge bases.

The remainder of this thesis moves from reasoning to reading; instead of inferring missing facts in a knowledge base, we apply the methods introduced in this chapter to models of natural language.

Chapter 4

Relation Extraction

4.1 Introduction

This chapter deals with the problem of *relation extraction*, a subtask of the general problem of natural language understanding. In relation extraction, we are tasked with deciding which relation from a formal knowledge base is expressed by a particular sentence, if any. For example, in the sentence "Barack Obama, the president of the United States, is in France today for a conference", our goal might be to predict that the relationship PRESIDENTOF holds between "Barack Obama" and "the United States". We can formalize this task as follows. Given a knowledge base $\mathcal{K}$ (containing a set of entities $\mathcal{E}$, a set of relations $\mathcal{R}$, and a set of triples (e_s, r, e_t) expressing instances of those relations), a sentence of natural language text, and two entities from $\mathcal{E}$ in the sentence, predict which relation from $\mathcal{R}$ holds between the two entities in the sentence, if any. To train a model on this task, we are also given a corpus of text; methods for using the text to train a model are described later in this chapter.

The work we present in this chapter is inspired by the methods discussed in Chapter 3. In Chapter 3, we developed methods for reasoning over knowledge graphs; in this chapter and the next, we show how those same methods can be applied to arbitrary models. That is, the main novelty of PRA and SFE is that they *generate feature matrices from graphs*; those feature matrices are useful for more than just link prediction in that graph. When the graph nodes correspond to elements of some other prediction model (such as KB entities in a relation extraction model, or nouns in a prepositional phrase attachment model, etc.), the PRA and SFE feature matrices can be used to augment that model.

It has long been understood that a large collection of common-sense knowledge is an essential component of any system that will be able to fully understand natural language. The Cyc project (Lenat, 1995) is one of the more well-known projects seeking to collect such common-sense knowledge, and the recent Winograd schema challenge (Levesque et al., 2012b) makes it very clear how necessary this kind of knowledge is in natural language understanding. Because of this necessity, there has recently been a surge of interest in constructing very large knowledge bases (KBs), such as Freebase (Bollacker et al., 2008), NELL (Mitchell et al., 2015), and DBPedia (Mendes et al., 2012), which contain facts about people, things, and places in the world. These knowledge bases have found use in providing training data for relation extraction and semantic parsing models (Krishnamurthy and Mitchell, 2012; Riedel et al., 2010), and forming the foundation of question answering systems such as Google's search and Apple's Siri.

One place where these knowledge bases have not typically found use is in lower-level natural language models. While it is now fairly well understood how to parse a question into an executable query over a structured knowledge base, it is less clear how to tractably incorporate a knowledge base into a model for parsing or part-of-speech tagging. In this chapter, we take a step in that direction by presenting a general technique for incorporating features generated from a knowledge base into *any* model that contains factors involving KB entities.

As an initial proof-of-concept of this idea, we add a PRA feature matrix as a factor in an off-the-shelf distantly-supervised relation extraction model, the MultiR system of Hoffmann et al. (2011). We chose this model for our initial experiment because the choice of KB is obvious (as they use a KB to obtain distantly-supervised training examples), and deciding how to incorporate a PRA factor is not difficult. In contrast, finding a suitable KB for a parsing model, as well as adding a factor in a way that still allows for efficient inference, are much more challenging (e.g., something like ConceptNet (Liu and Singh, 2004) could be used, though its coverage may be questionable). However, the gains we achieve by adding PRA features into the MultiR model are quite substantial, and we believe they provide sufficient motivation to explore incorporating these same ideas into more low-level NLP models.

In the remainder of this chapter, we first briefly describe the PRA and MultiR models, as well as other related work, then we describe how we incorporate PRA features into MultiR. We then present an experimental comparison between our PRA-augmented MultiR model with various techniques from prior work, showing that our method substantially outperforms prior techniques in both aggregate and sentential relation extraction.

4.2 Prior work relevant to this chapter

In this section we briefly remind the reader of how PRA works, introduce the model into which we inject a PRA feature matrix, and describe a few other pieces of work that are closely related to what we present in this chapter.

4.2.1 The Path Ranking Algorithm

We discussed PRA in detail in Section 2.2.2. Here we simply remind the reader that PRA computes a feature matrix over node pairs in a graph. Prior uses of PRA have all been to perform link prediction in that graph, for a task such as knowledge base completion. The feature matrices computed by PRA, however, are more generally applicable than just to the task of link prediction. In this chapter, we inject a (binarized) PRA feature matrix into a more complex graphical model, the MultiR model for performing relation extraction, described in the next section.

4.2.2 MultiR

The MultiR algorithm (Hoffmann et al., 2011) is a probabilistic model for performing relation extraction with distant supervision. The model, shown in Figure 4.1a, is *distantly supervised* because, while it makes predictions on individual sentences, it has no annotated sentences as training data. The input to MultiR is a KB and a set of sentences that each mention a pair of KB entities. The model assumes that at least one of the sentences expresses each relationship seen between the entity pair in the KB, and learns a model to predict which sentences express which relations. In the model description in Figure 4.1a, S is the set of sentences between any two entities e_1, $e_2 \in$ E and the Z_i's are the latent variables that range over all relation names and a distinct value NONE. The model outputs a binary relation vector Y, with indicator bits for each relation. It is trained using an online perceptron-style algorithm.

4.2.3 Other Related Work

We briefly mention here a few other pieces of related work, particularly those that we will compare our work against. Surdeanu et al. (2012) presented MIML-RE, which is very similar to the MultiR algorithm, but relaxes some of the hard constraints used by MultiR. Weston et al. (2013) is the work that is most similar to ours, in that they also attempted to add information from a knowledge base to a relation extraction model. While our technique includes symbolic information from the KB as features in the MultiR model (see Section 4.3, Weston's method uses a vector space embedding of both the KB and the individual sentences to try to improve relation extraction performance. We compare against both MIML-RE and the work of Weston et al. in Section 4.4.

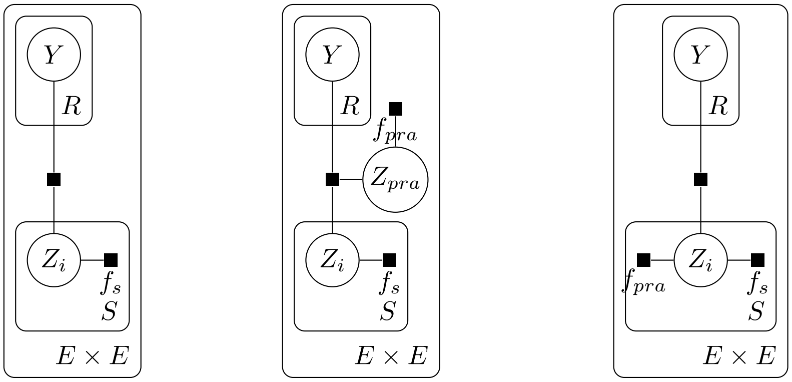
Additionally, Nakashole and Mitchell (2015) show a method for incorporating background knowledge into predicting prepositional phrase attachment. Their work provides additional motivation for this line of research, as we believe that the addition of a PRA-style feature matrix from a suitable KB such as ConceptNet (Liu and Singh, 2004) would further improve their results.

4.3 PRA-Augmented MultiR

There are two options for adding a PRA factor into the MultiR model. The first (and more natural) place to add the factor is once per entity pair, as an additional component to the factor connecting the Y and Z_i variables. This is shown in Figure 4.1b. The issue with this placement is that the factor connecting Y and Z_i is a deterministic OR, and the model is trained using an at-least-once assumption; this means that if the model can explain the existence of a relationship between an entity pair with the PRA factor, it will tend to not learn from the sentences for that entity pair.[1]

An alternate placement is to include the PRA features as an additional component of every mention of the entity pair, shown in Figure 4.1c. One way this can be thought of is as a stronger kind of supervision; the stronger an entity pair exhibits a particular relationship, as determined by the PRA features, the more likely the dependency parse features of the MultiR model are to be up-weighted for that sentence. Another way to view this placement is as a modification of the “expressed at least once” assumption made by the deterministic OR factor in MultiR. Adding PRA features for each mention has a similar effect to gradually changing the “expressed at least once” assumption to an “expressed every time” assumption the more canonical the entity pair is according to the PRA features. Work by Betteridge et al. (2014) found that strengthening the “expressed at least once” assumption can be helpful, and the effectiveness of this feature (seen in Section 4.4) placement also corroborates that result. When judging aggregate performance, this also gives the model access to a much richer set of features than just those expressed in the corpus.

[1] Changing the factor connecting the Y and Z_i variables to something other than a determinstic OR could potentially mitigate this issue. However, this would greatly complicate the inference algorithm developed for MultiR, and we did not experiment with this.



(a) MultiR (b) Per Entity-pair (c) Per Mention

Figure 4.1: Plate notation of the MultiR model and our modifications of it. R is the number of relations in the KB, E is the number of entities, and each Y variable encodes whether an entity pair expresses relation R. There is one Z variable per sentence, encoding which relation is expressed by that sentence. The Per Entity-pair extension adds a PRA factor once for every entity pair; the Per Mention model instead adds a PRA factor once for every *mention* of every entity pair.

4.4 Experiments

To evaluate the performance of this model, we use the dataset developed by Riedel et al. (2010), which aligns Freebase relations with the New York Times corpus. The dataset contains a number of sentence mentions for each entity pair. The entities are extracted from sentences by Stanford Named Entity Tagger (Finkel et al., 2005), and they participate in 52 separate relations from Freebase. The sentence level features include the part of speech tags, named entities and dependency tree paths. The evaluation protocol sorts all predictions for all relations by confidence and computes a precision-recall curve over those predictions. We evaluate extraction performance in aggregate (comparing predicted Y values to truth from the KB), and at the sentence level (comparing predicted Z_i values to human-annotated Z_i values in the test set).

To extract PRA features, we used a dump of the Freebase graph, excluding relations that were not relevant to our prediction task, such as the relations under /USER and /COMMON, and relations dealing with things like individual music tracks and TV episodes). We removed any direct edges between entity pairs in the training and test data, then ran the first step of PRA to produce a binary feature matrix. We then added this matrix as a factor to the MultiR model in the two ways described in Section 4.3. We found that because this new factor increased the size of the feature space, additional regularization was needed. We added an L2 regularization parameter to the model, which we tuned using 5-fold cross validation on the training data. This significantly improved the performance of our augmented model, though a similar addition to MultiR did not improve performance, so we did not include it in our final experiments.

We first present a comparison of aggregate extraction performance between the ways of augmenting MultiR with PRA features. Figure 4.2 shows the results of the two different augmentations. We also include here two simple baselines for comparison. "PRA Only" shows the result when just using PRA features in the model (i.e., assuming that there are no sentence mentions for any entity pair). "Interpolated" is the result of a weighted interpolation between "PRA Only" and MultiR, showing that joint training between the PRA features and MultiR's dependency path features is crucial, as the interpolated result is quite poor.

Next, Figure 4.3 shows a comparison between the best PRA-augmented model ("PRA Per Mention" in Figure 4.2) and prior work on aggregate extraction performance. As can be seen from the figure, our method substantially outperforms all prior work, including models that just used dependency path features (MultiR and MIML-RE) and models that combined KB embeddings with dependency path features (Weston). It appears that the embedding method vastly under-utilizes the knowledge base, as the same information is available to both Weston's method and ours. We believe this is because much of the information contained in the knowledge base is not amenable to low-rank representations (Nickel et al., 2014), and so our method that incorporates symbolic information from the KB can get much more leverage from that information. Weston's method also is a kind of interpolation between a KB embedding model and a sentence-level embedding model, which, judging by our interpolated result with PRA features,

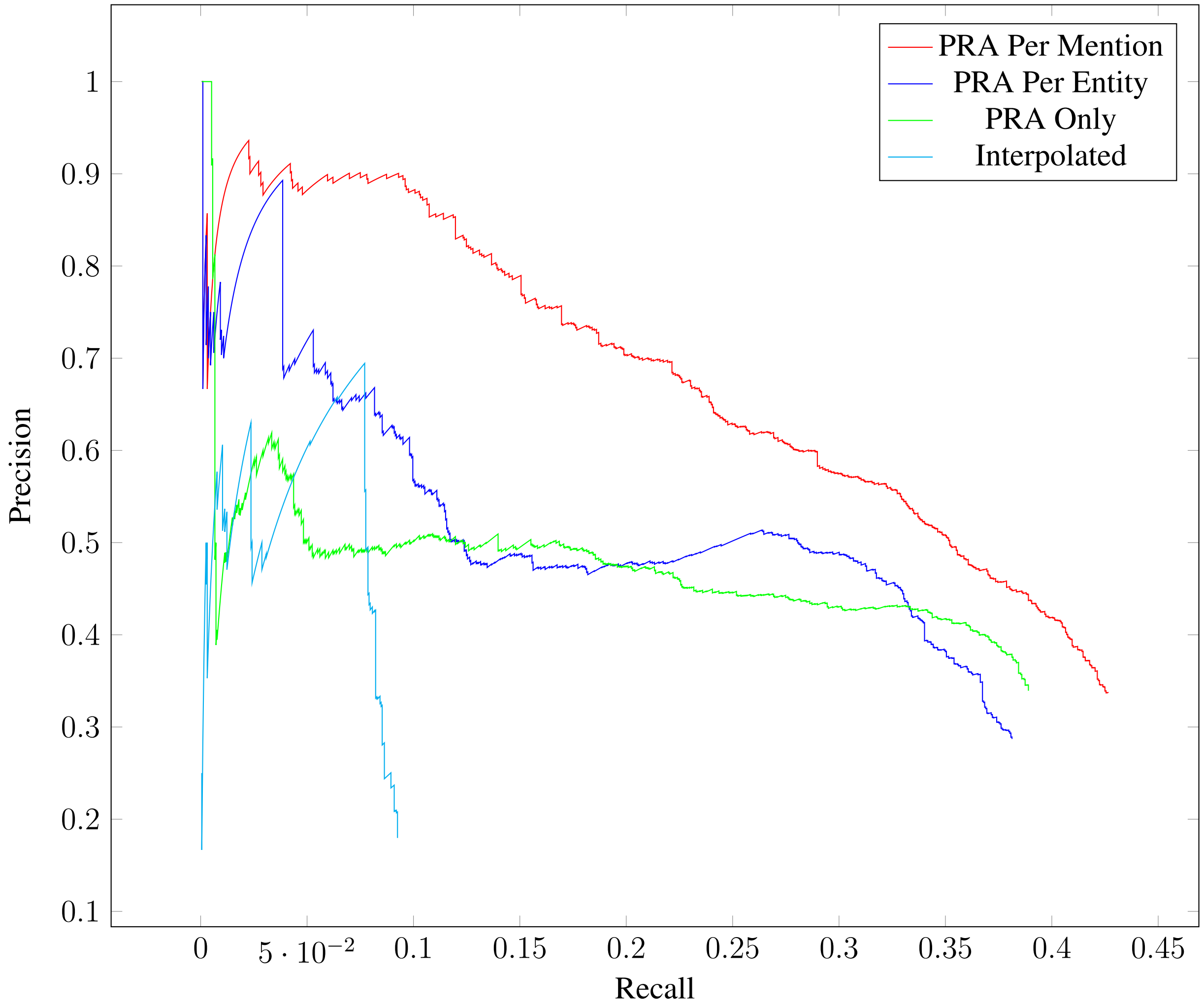


Figure 4.2: Aggregate extraction performance of the augmented MultiR methods introduced in this chapter, along with two baselines.

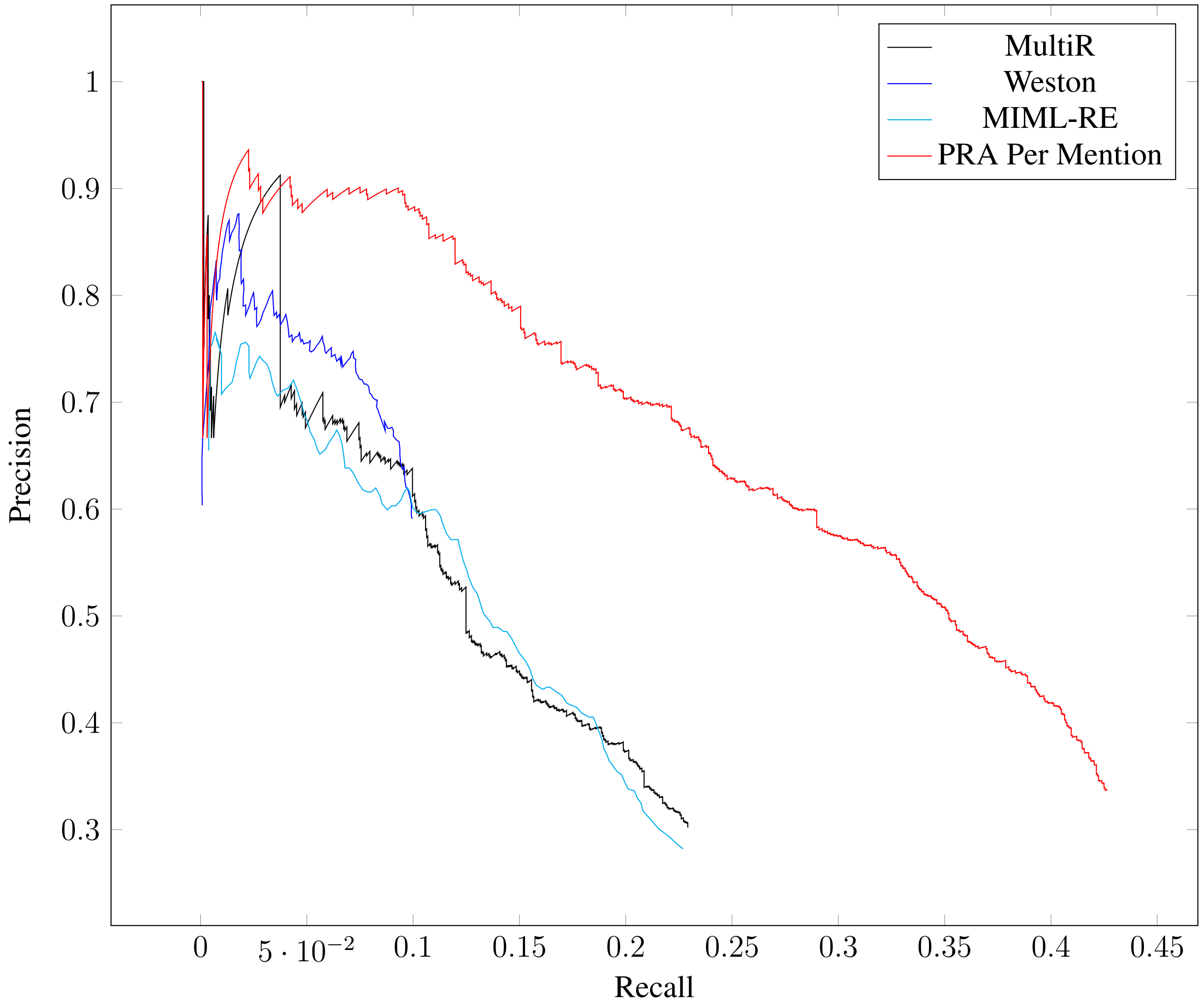


Figure 4.3: Aggregate extraction performance of our method compared with several methods from prior work.

may explain their poor utilization of the KB.

Lastly, we present a comparison of sentential extraction performance between MultiR and our augmented version. We note here that the PRA features we added are aggregate-level features, not sentence-level features. A relation may be true in aggregate but not expressed by a particular sentence, and so it is possible that the PRA features could mislead the model when trying to infer whether that relation is expressed by a specific sentence. Because the PRA features could be misleading at test time, we also experimented with only using the PRA features at training

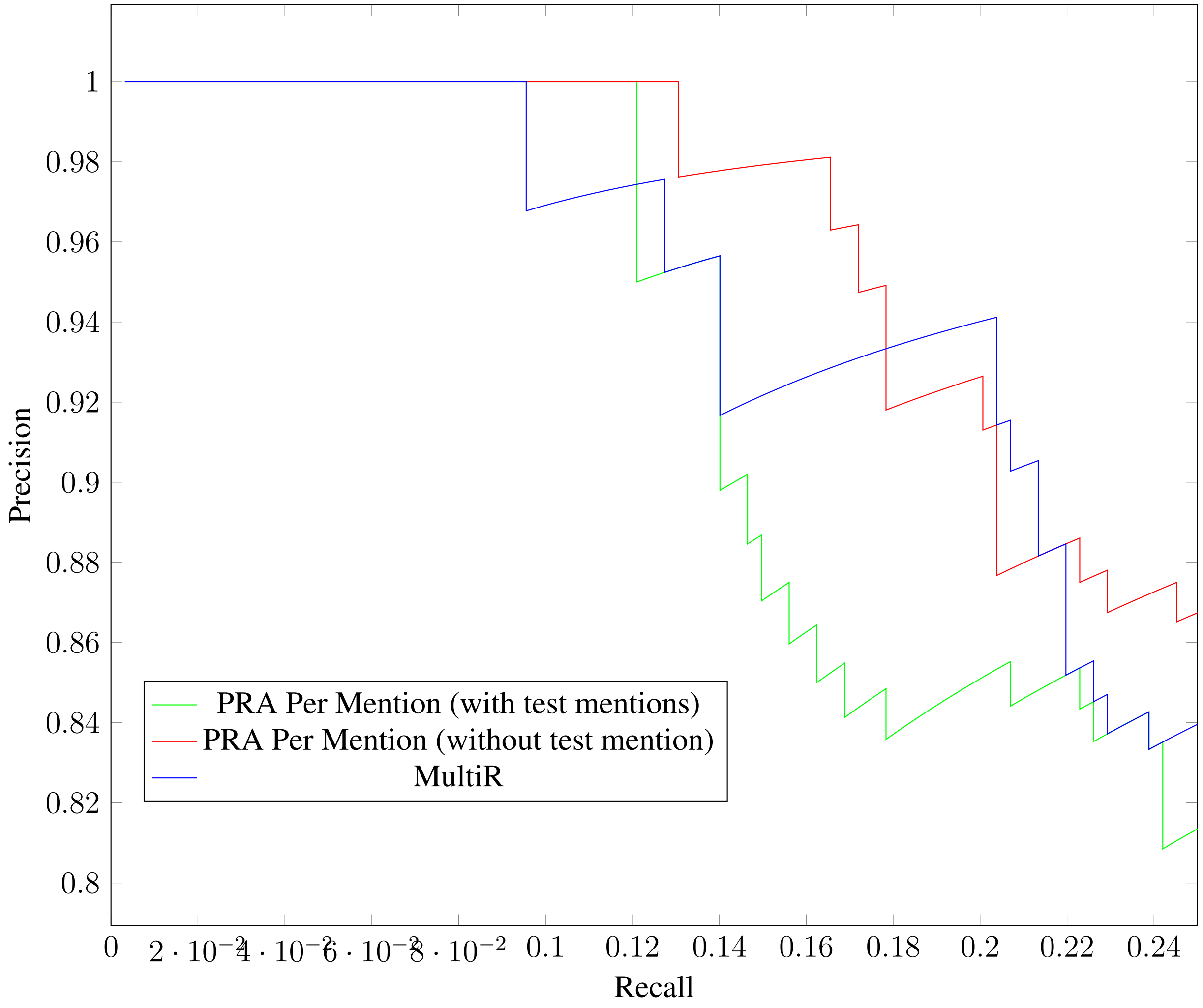


Figure 4.4: Sentential extraction performance of our method compared with MultiR.

time, removing them during testing. Figure 4.4 shows the results of this comparison. PRA-augmented MultiR performs slightly worse than MultiR, while, somewhat surprisingly, using PRA features only at training time performs slightly better. We believe this improvement is because the presence of the PRA features at training time gives the model a stronger and better supervision signal, nudging the model to learn more from entity pairs that are strongly associated with a relation than from those that are not, resulting in better learned weights for the dependency features.

4.5 Conclusion

We have addressed the problem of incorporating features from a knowledge base into low-level natural language processing models. We introduced a simple technique that computes a PRA feature matrix from a knowledge base and incorporates it as a factor in arbitrary graphical models. As an initial proof-of-concept of this general technique, we showed dramatic improvements when using this idea to augment an off-the-shelf relation extraction model, MultiR. We believe this result amply shows the utility of this technique, and hope to similarly apply it to other low-level NLP models in the future.

Chapter 5

Modeling Lexical Semantics

We saw in Chapter 3 that graph-based reasoning techniques can successfully be used to model the semantics of formal knowledge base relations. That is, given a set of training instances for a KB relation, we can learn models that do a reasonably good job of predicting whether two entities participate in that relation.

In this chapter we explore whether we can also learn models of *lexical* semantics, a key component of any reading system. Instead of trying to model formal knowledge base predicates, which generally have well-defined semantics, this chapter explores models of lexical predicates, which have much less well-defined semantics. Verb phrases can have many different meanings in different contexts, and can overlap in meaning with many other verb phrases. It is not clear a priori that the graph-based reasoning techniques used for formal KB predicates can also successfully model lexical predicates; formal logic has not generally been able to model language very well. The experiments in this chapter show, however, that PRA and SFE (discussed in Chapter 3) are also fairly successful in modeling lexical semantics, at least in a few limited settings.

We test the ability of PRA and SFE to model lexical semantics in the context of two question answering tasks. We choose tasks with simple questions where there is some hope of getting a correct answer with just a lexical model; more complex questions require compositional models of sentential semantics. While the end goal is to use these lexical models as pieces in a compositional sentential model, we must first demonstrate that these models can adequately capture lexical semantics, and that is the focus of this chapter.

Section 5.1 applies a PRA model to a small set of multiple choice science questions. These questions require lexical models that can successfully reason over background knowledge to score candidate answers. In contrast, Section 5.2 addresses simple reading comprehension questions, which require an understanding of discourse semantics but not background knowledge.

With both of these datasets, the formal definition of the problem we are trying to solve is, given some amount of background text (either a collection of science texts, or a short passage describing a situation) and a question, produce the correct answer to the question (either as a choice among several options, or as a word of natural language text). To make this problem more approachable, however, we will reduce the question answering task to a problem of link prediction in a specially-constructed graph, allowing us to directly use the methods developed in Chapter 3. Particularly, we will seek to score lexical predicates with their arguments, such as IS_IN("John", "playground"), and COMPARE("balance", "mass") (attempting to answer whether John is in the playground, or whether a balance can be used to compare mass, respectively). We will construct a graph such that "John" and "playground" are nodes, and we wish to know if there should exist an edge of type IS_IN between these two nodes.

5.1 Multiple choice science questions

The content in this section is derived in part from a paper titled "Combining Vector Space Embeddings with Symbolic Logical Inference over Open-Domain Text" (Gardner et al., 2015).

When building models of lexical semantics, the aim is to score potential arguments for predicative words such that "correct" arguments have higher scores than "incorrect" arguments. For instance, a model for the verb "eats" should give a higher score to the argument pair ("a bear", "a fish") than to the argument pair ("a fish", "a bird"). For many such predicative words, including "eats", a model that is able to correctly score these arguments must incorporate some amount of background knowledge.

Multiple choice science questions provide a nice test case for these kinds of lexical models. Because they are multiple choice questions, the argument pairs that need to be ranked are given, so an evaluation can focus on the lexical model itself instead of on a search procedure to generate candidate answers. And because the domain is science questions, most of the necessary lexical models require common sense background knowledge, so the questions are a good measure of how well a lexical model captures background information. The challenge with these questions is that most of them require more than a simple lexical model to arrive at a correct answer; we will have to carefully select questions that are complex enough to be interesting, but simple enough to not need complex sentential semantics.

In this section we present an application of the path ranking algorithm (PRA; see Section 2.2.2) to the task of modeling lexical semantics, evaluated on a small set of multiple choice science questions. We show that we are able to correctly answer 67% of the questions in our test set, on par with the state-of-the-art Aristo 1.0 system (Clark et al., 2013).[1] The remainder of this section first describes the questions and background data used in this task (Section 5.1.1), then describes how we applied PRA to this problem (Section 5.1.2). Finally, Section 5.1.3 shows and discusses the results of our experiements.

5.1.1 Data

The questions we experimented with come from the New York Regents Exam (Clark et al., 2013). We hand selected 24 of these questions that we believed could be answered solely by a lexical model. A few example questions follow:

> The functions of a plant's roots are to support the plant and
>
> (A) make food
>
> (B) produce fruit

[1] These questions all have 4 possible choices, so random guessing yields 25% accuracy.

(C) take in water and nutrients

(D) aid in germination

Which tool should a student use to compare the masses of two small rocks?

(A) balance

(B) hand lens

(C) ruler

(D) measuring cup

Which characteristic can a human offspring inherit?

(A) facial scar

(B) blue eyes

(C) long hair

(D) broken leg

Sleet, rain, snow, and hail are forms of

(A) erosion

(B) evaporation

(C) groundwater

(D) precipitation

Examples of the more complicated questions that we omitted include, “In New York State, the longest period of daylight occurs during which month?”, and “Water freezing is an example of a (A) liquid changing to a solid”. Superlatives and higher-order relations are not easily captured with simple lexical models.

The lexical information necessary to answer the questions in this data set include models for the verb phrases “make”, “produce”, “take in”, “compare”, “inherit”, and “are forms of”. There are only 24 questions in the data, however; this is not nearly enough to learn good models of lexical semantics. To obtain data to learn these models, we turn to collections of scientific background knowledge. The texts that we used were those used by the Aristo system (Clark et al., 2013) (including the Barrons 4th Grade Science Study Guide, the CK12 Biology Textbook, and the science section of Simple English Wikipedia; below referred to as the "Aristo" extractions), and a subset of the SVO triples found in the ClueWeb corpus (Talukdar et al., 2012) that were deemed relevant to the task (the selection process is explained in more detail in Section 5.1.2; this data is referred to below as the "ClueWeb" extractions). After performing the processing described in the next section, we ended up with 532k triples in the Aristo extractions, and 1,830k triples in the ClueWeb extractions.

5.1.2 Methods

PRA performs link prediction in a graph (see Section 2.2.2). Here we describe how we cast the multiple choice question answering problem into a form that can be used with PRA. At a high level, our method takes the background texts described in the previous section and converts them into a graph where the nodes correspond to noun phrases and the edges correspond to verb phrases. Each question is then converted into a set of queries on this graph, which are scored using PRA. In the remainder of this section we describe this process in more detail.

Forming a graph from the background texts is fairly straightforward. We first perform a simple subject-verb-object (SVO) extraction to obtain triples from the text. This was already done for the ClueWeb corpus; for the Aristo corpora, we used OLLIE, an open information extraction system (Mausam et al., 2012). Given a set of triples, the graph was constructed as described in Section 2.1.2—a node is created for each unique noun phrase, and for every (s, v, o) triple, an edge of type v is added between the nodes corresponding to s and o.

We used a similar process to construct queries over this graph from the questions. Each question-answer pair was converted into a declarative sentence (e.g., "A student should use a balance to compare the masses of two small rocks.", for the second question listed above). We then do a similar SVO extraction on this declarative sentence, and keep any triple containing words from the answer. In order to have a good set of queries on this small data set, we performed this process by hand, while also removing light verbs such as "use" (so, e.g., "use to compare" became "compare"). The triples corresponding to the first question above are as follows:

(A) (roots, make, food)

(B) (roots, produce, fruit)

(C) (roots, take in, water), (roots, take in, nutrients)

(D) (roots, aid in, germination)

Answering this question thus requires for lexical models, for "make", "produce", "take in", and "aid in". We use PRA over the background texts to learn these models. That is, we take all edges in the background graph that have a type matching the verb phrase; i.e., the training data for the PRA model for "make" (used to score the triple (roots, make, food)) consisted of all (subject, verb phrase, object) triples where the verb phrase was "make". We subsampled these triples until we had at most 3,500 examples per relation. Using this training data, we then learned a PRA model (as described in Section 2.2.2) and scored all of the triples for each candidate answer to each question, taking the answer that had the highest combined score.

Our initial attempt used the triples in the questions and in the extractions exactly as found. However, as each noun phrase is represented as a node in the graph, long noun phrases in the triples made for a very disconnected graph, and PRA was not able to successfully score very many of the triples in the questions (e.g., "the masses of two small rocks", in the second example question above, had no connecting edges in the graph, and thus PRA could not give a score other than 0 for queries involving it. See Figure 5.1). We tried to fix this issue by including edges between noun phrases: an edge from each noun phrase node to the node representing its head, edges from single-word noun phrases to their lemmas, and other similar edges (see Figure 5.2). However, this made the graph so densely connected that random walk inference was intractable—the computation required to find connections by random walks is $O(\text{degree}^{\text{pathlength}})$, and adding

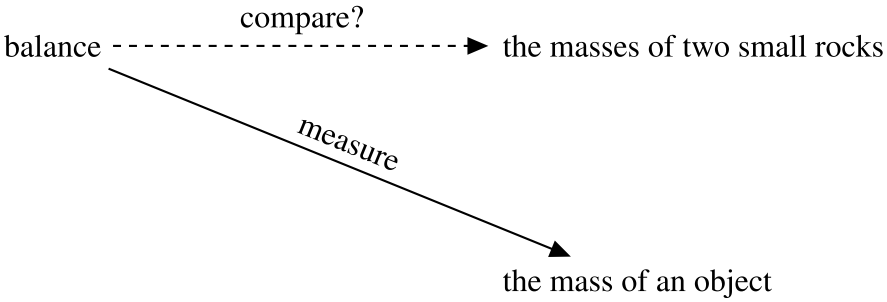


Figure 5.1: An example graph showing the futility of including complex noun phrases as nodes in the graph. There are no connections to the node "the masses of two small rocks", because it was only seen once, in the test query.

edges in this way both dramatically increased the average degree of each node and significantly increased the path length between noun phrases.

To solve this problem, we processed all of the triples (both from the questions and from the extractions) to only keep the lemmatized head of each noun phrase, and the lemmatized verb phrase (dropping any auxiliary or modal verbs, but keeping prepositions). For example, the triple (an offspring, can inherit, blue eyes) became (offspring, inherit, eye). This naively throws away a lot of information, some of which may be very important to answering the question (e.g., "scratched eyes" and "blue eyes" have the same simplification, though only one can be inherited). However, some method for making random walk inference both possible and tractable was necessary, and this naive approach gave us a starting place that actually had decent performance. A representation of this final graph is shown in Figure 5.3.

5.1.3 Experiments

The experiments we present involved using PRA with a graph constructed from the Aristo and ClueWeb extractions, as described in Section 5.1.2. For comparison, we include the Aristo system (Clark et al., 2013) and a naive baseline that scored each question triple by returning the number of its occurrences in the set of extractions.

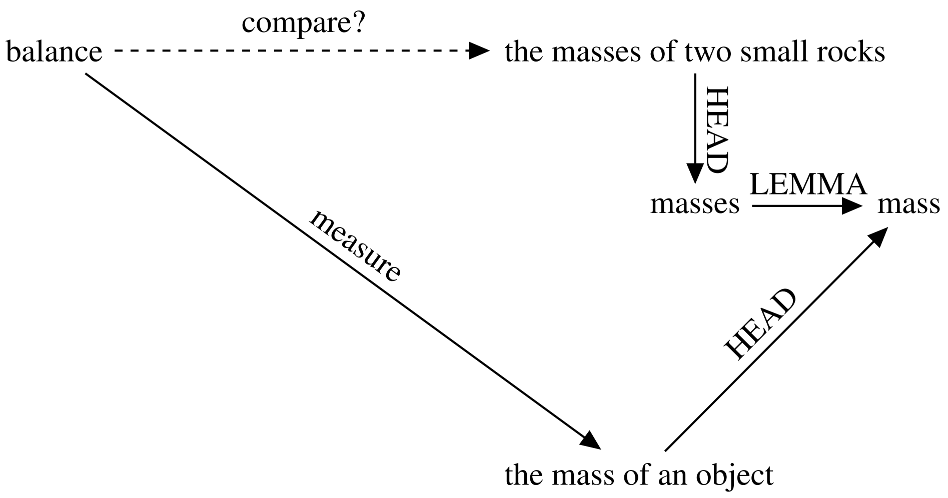


Figure 5.2: Adding HEAD and LEMMA edges improves the connectivity of the graph, allowing for paths to actually connect the source and target nodes of the query. However, because there are thousands of sentences involving the word “mass” (not shown in this example), the outdegree at the nodes for “mass” and “masses” becomes too large to permit inference using random walks.

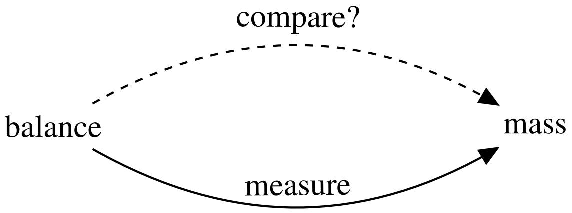


Figure 5.3: The final form of the graph we used. All complex noun phrases are replaced by their lemmatized heads. While this loses a lot of potentially useful information, the graph is now connected enough (and has low enough degree) that inference is possible.

Method	Correct	Precision	Recall
Best PRA result	16	0.70	0.67
Aristo	16	0.70	0.67
Baseline	5	0.83	0.21

Table 5.1: Number of correct answers, precision, and recall of the systems we experimented with, on our small test set of 24 questions.

Table 5.1 shows the results of our experiments. Our best PRA model had performance that was equivalent to that obtained by Aristo. We want to emphasize here that there was a lot of variance in our experiments—some runs of the system got as low as 11 correct, and attempts to use vector space random walks or additional edges from curated knowledge bases like NELL, Freebase or ConceptNet proved inconclusive. We need a larger test set to be confident in making a fair comparison between methods on this task. Our intent in presenting these results is simply to show that graph-based reasoning techniques seem to capture lexical semantics reasonably well. We believe that smarter approaches to training data selection and graph construction, in addition to incorporating these lexical models into a more complex question answering system, could give performance better than the current state-of-the-art.

In the remainder of this section we give some analysis of the successes and failures of PRA on this task. First, we show some interesting Horn clauses that were assigned high weight by PRA; the algorithm is frequently able to discover important aspects of the semantics of textual relations. Some examples for the relation “depend on” are the following:

- <“depend on”, “depend on”> (“depend on” is transitive)
- <“use”, “depend on”>, and <“use”, “give”$^{-1}$> (if I use something, I depend on its dependencies, or something that gives it)
- <“depend on”, “make”> (if I depend on something, it might be because I depend on something it makes)

Next, we discuss some specific examples from the test set. First, note that the baseline (which simply looked up triples in the KB), only gave an answer for 6 of the questions. On 5 of them it was right, so the baseline has a high precision, but PRA is able to correctly answer many more questions because it can infer facts that are not present in the KB (e.g., (rain, is, precipitation) never occurred in the data, but (precipitation, consists of, rain) did, and PRA used that fact to correctly answer one of the questions). Interestingly, PRA correctly answered the question that the baseline gave a wrong answer for (the first example question given above), because even though the triples (root, make, food), and (root, produce, fruit) occurred several times in the extracted data, the learned PRA models assigned them low probability.

One weakness in our approach can be seen in the following question: “A decomposer is an organism that (A) hunts and eats animals (B) migrates for the winter (C) breaks down dead plants and animals (D) uses water and sunlight to make food”. The correct answer is C, but PRA gives B for the answer because (?, migrates for, winter), with any subject, has a high probability in the model for “migrates for” (if not for that, PRA would have gotten the question correct). A significant portion of the precision errors made by PRA would be solved with a better method for comparing scores across relations, especially where one of the arguments to the relation has an exceptionally high probability.

Finally, a very significant weakness with the approach we took has already been mentioned: only keeping the head of each noun phrase can throw away information that is critical to answering the question. Between “metal fork” and “plastic spoon”, it is clear which is the better conductor of electricity, for example, but given only the heads “fork” and “spoon”, there is no way to reliably obtain the correct answer. We did this stemming to decrease the computational requirements of PRA; our approach is essentially equivalent to clustering the nodes in the KB graph, shortening the connecting paths between noun phrases and decreasing the average degree of the graph. Another approach that does not throw away potentially critical information would be to prune the graph, keeping only edges that are deemed relevant to solving a particular query. This kind of “micro-KB” construction would also decrease the average degree of the graph and allow for deeper inferences while keeping the computational requirements of PRA to a manageable level.

5.2 Reading comprehension questions

Having shown that graph-based reasoning can successfully incorporate background information into models of lexical semantics, in this section we turn to the task of reading comprehension, with a dataset that requires incorporating discourse information into lexical models, instead of background knowledge.

5.2.1 Introduction

Reading comprehension is the central task of natural language understanding, and it has seen a surge of interest in recent years (Levesque et al., 2012a; Richardson et al., 2013; Weston et al., 2015). In order to comprehend a passage of text, a machine reader must necessarily incorporate models of lexical, sentential, and discourse semantics. Lexical semantic models allow the reader to understand the words used in the passage; sentential semantic models let the reader compose the lexical models of the words in a sentence into a cohesive representation of their meaning; and discourse models bridge the sentences in the passage to find the meaning of the passage as a whole.

While state-of-the-art systems are a long way away from general, broad coverage reading comprehension, bAbI (Weston et al., 2015), a recently released synthetic dataset from Facebook AI Research, allows us to build systems that can capture relatively simple phenomena that are frequently seen in real text. This dataset has passages that are several sentences long, where each sentence has very simple syntax and semantics. The questions require a reader to put together information from one to three of the sentences in the passage. For instance, to answer “Where is

John?", the reader must find the sentence that last mentions John's location; to answer "Where is the football?", the reader must find who last had the football and their last stated location.

In this work we are interested in models of lexical semantics that incorporate discourse information. That is, given a question like "Where is John?", we seek a model for the predicate IS that depends on the discourse, such that we can score IS("John", x), for various values of x that appear in the passage. The lexical component of these models should capture the fact that a person *is* where they have *moved to* or *traveled to*, for example, and the discourse component should capture the fact that a person *is* only in the *most recent* location that they have moved to.

We present a simple method that models these lexical and discourse phenomena jointly. Our method represents the passage of text as a graph, then learns models of two-argument predicates by finding paths through the graph that connect the two arguments. Because of the way the graph is constructed, these paths contain both lexical and discourse information, and because we have training data in the bAbI dataset, we can learn discriminative models to score predicates with their arguments.

We evaluate this model on the bAbI reading comprehension questions. On most of the question types that only require lexical and simple discourse modeling, our method achieves perfect or near-perfect accuracy. For completeness, we also evaluate the method on question types with more complex sentential semantics, though we do not attempt to model these phenomena in this work. Even with our simple lexical model, however, we achieve an average of 82% accuracy on the 18 question types we attempted.

5.2.2 bAbI

The bAbI dataset was developed as a set of synthetic toy tasks that are prerequisite to any system that can read and comprehend real text (Weston et al., 2015). The data has 20 question types, each of which focuses on a particular language phenomenon. Each question type has 1000 training questions and 1000 test questions. We do not have space to give a complete description of all 20 question types in the bAbI dataset; for this we refer the reader to the original dataset description. We do give examples of some of the question types, however (examples are taken from the original paper).

The first, and simplest, question type requires finding one sentence in the passage that contains the answer. An example is the following:

> John is in the playground.
>
> Bob is in the office.
>
> Where is John? A: playground

A more complicated question type, requiring three supporting sentences, is this:

> John picked up the apple.
>
> John went to the office.
>
> John went to the kitchen.
>
> John dropped the apple.
>
> Where was the apple before the kitchen? A: office

Some question types require simple coreference resolution:

> Daniel was in the kitchen.
>
> Then he went to the studio.
>
> Sandra was in the office.
>
> Where is Daniel? A: studio

Others ask questions that require positional reasoning, reasoning about the size of objects, reasoning about time, and other relatively simple phenomena. Most of the question types use the same wording for all questions, though some of them have several different wordings within the same question type. For example, the question type dealing with sizes has questions like this:

The suitcase is bigger than the chocolate.

The box of chocolates fits inside the chest.

The suitcase fits inside the chest.

Is the chest bigger than the box of chocolates? A: yes

Does the chocolate fit in the chest? A: yes

5.2.3 Method

Our method for building models of lexical and discourse semantics has two main parts: we first convert the passage (along with the question and candidate answer) into a graph, and then we score (question, answer) pairs using features generated from that graph.

To construct a graph from a passage of text, we first dependency parse the sentences in the text, and dependency edges between nodes representing words form the bulk of the graph we construct. We number the sentences, giving separate nodes in the graph for each word in each sentence, and we connect subsequent occurrences of the same word across sentences. That is, if the word "John" appears in the first sentence and the third sentence, there will be an edge between the two "John" nodes indicating that the node "3:John" is the next occurrence of "John" after "1:John".[2] The label we give to this edge (seen in Figure 5.4) is *last instance*. We additionally add an edge of type *instance* from each instance of a noun phrase in the question sentence to all prior occurrences of that noun phrase.

To add a question with a candidate answer to the graph, we first convert the (question, answer) pair to a declarative sentence. With the simple kinds of questions in the bAbI dataset, this only requires undoing wh-movement and replacing the wh-phrase in the question with the candidate answer. For example, the question "Where is John?" with the answer "playground" would be

[2] Because of some inconsistency in the capitalization in some of the bAbI questions, we also lower case all words before creating nodes for them.

converted to “John is in the playground.”[3] This sentence is processed and added to the graph exactly as if it were the last sentence in the passage.

To get candidate answers for each question, we extract all nouns and adjectives from the passage.[4]

Finally, we select two words from the question sentence to use as arguments for a predicate. Our method for doing this is heuristic, but many of the question types in the bAbI data have only two nouns and a single predicate, which gives us only one reasonable choice anyway. This is actually very helpful for our purposes, because it allows us to experiment with joint lexical-discourse models without a lot of additional effort to semantically parse the sentence. In the question-converted-to-sentence “John is in the playground”, we would select “John” and “playground” as the arguments, and learn a lexical-discourse model for the predicate IS IN.

The model for each predicate is a discriminative logistic regression classifier that scores argument pairs. The features for an argument pair are drawn from the graph. We find paths through the discourse graph that connect the nodes corresponding to the argument pair, and use these paths as binary features in the logistic regression classifier. Specifically, we use subgraph feature extraction (SFE; see Section 3.4) with slightly modified PRA-style features. The only modification we make is that we include node labels as part of the feature description, when the node in the path corresponds to a verb in the sentence. This allows us to still capture lexical information while using dependency labels as edges in the graph. See Figure 5.4 for an example graph and extracted features.

As an example of the kinds of features learned by these models, a highly weighted feature for the first question type is <*last instance*, *nsubj*, “went”, *prep_to*$^{-1}$, *instance*$^{-1}$>. This path

[3] While this should be straightforward to do algorithmically, in this work we hand-coded this transformation for each of the small number of question types in the bAbI data. Also, yes/no questions have no wh-phrase, and in this case we simply undo movement to obtain a declarative sentence.

[4] For one of the question types, we also add the special value “nothing”, which is a common answer but never appears in the passage.

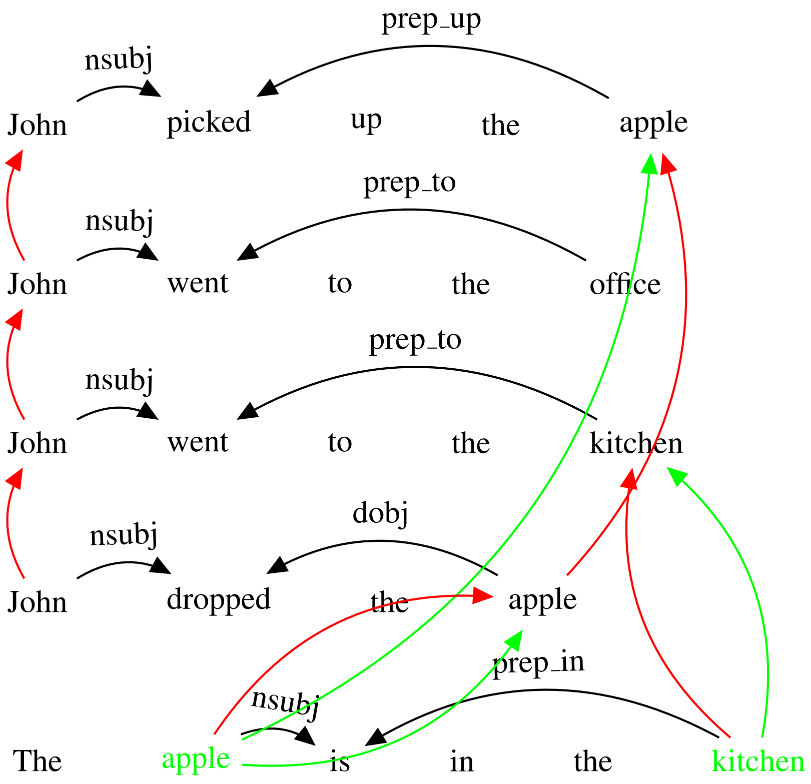


Figure 5.4: An example graph constructed from a question in the bAbI dataset. The green nodes ("apple" and "kitchen" in the last sentence) are the query nodes. The red edges are of type *last instance*, and the green edges are of type *instance*, with the labels omitted in the graph to improve readability. One example feature that can be found from this graph is <*last instance*, *dobj*, "dropped", *nsubj*$^{-1}$, *last instance*, *nsubj*, "went", *prep_to*$^{-1}$, *instance*$^{-1}$ >. The *instance* and *last instance* parts of this path encode discourse information, while the dependency edges and verbs encode lexical information. Intuitively, the path feature finds the sentence stating that an item (the apple) was dropped, finds the person who dropped it (John), then looks for the last place that person was before dropping the item (the kitchen).

connects the query node (e.g., “John”) with the candidate answer (e.g., “playground”) in the discourse graph. The path goes from “John” to the most recent prior occurrence of “John”, then follows a dependency path from “John” through the verb phrase “went to”, to the object of the verb phrase. *Instance* edges connect a query word with each occurrence of the word in the passage, so the final step in the path goes from the sentence mentioning John’s location to the query sentence. Intuitively, this feature finds the last time the person was mentioned, then looks for where he or she “went to”. The *instance* and *last instance* components of this path capture discourse phenomena, while the dependency edges and verbs capture lexical phenomena.

5.2.4 Experiments

We experimented with 18 of the 20 question types in the bAbI reading comprehension data set. We excluded two of the question types as needing a richer model of sentential semantics than our simple lexical model can provide. For each question type, we used the training data to learn a single model that scored candidate answers for each question, and we report accuracy on the test data.

For comparison, we include three baselines, all taken from the paper introducing the bAbI dataset (Weston et al., 2015). The first is an LSTM that is trained to predict the answer given the entire passage and the the question. The other two methods are an SVM model and a memory neural network (Weston et al., 2014). The performance of these last two methods is not directly comparable to our work, as they use an additional training signal. In addition to the passage, the question, and the answer, the bAbI dataset also contains which sentences from the passage are necessary to answer the question. We did not use this information in our work, while the SVM and the MemNN did. The LSTM is the best prior approach that uses the same training setting that we did.

[5] Due to an oversight, we treated this as a yes/no question type, with all “maybe” answers being considered “no” at both training and test time.

[6] The average is only over the 18 question types we attempted, so we do not unfairly advantage our average

Question Type	Our Accuracy	LSTM	SVM*	MemNN*
1: Single supporting fact	1.000	0.500	0.990	1.000
2: Two supporting facts	0.873	0.200	0.740	1.000
3: Three supporting facts	0.852	0.200	0.170	1.000
4: Two argument relations	1.000	0.610	0.980	1.000
5: Three argument relations	0.981	0.700	0.830	0.980
6: Yes/No questions	0.985	0.480	0.990	1.000
7: Counting	N/A	0.490	0.690	0.850
8: Lists / Sets	0.918	0.450	0.700	0.910
9: Simple Negation	0.999	0.640	1.000	1.000
10: Indefinite Knowledge[5]	0.999	0.440	0.990	0.980
11: Basic coreference	0.629	0.720	1.000	1.000
12: Conjunction	1.000	0.740	0.960	1.000
13: Compound coreference	0.765	0.940	0.990	1.000
14: Time manipulation	0.793	0.270	0.990	0.990
15: Basic deduction	0.335	0.210	0.960	1.000
16: Basic induction	0.997	0.230	0.240	1.000
17: Positional reasoning	0.859	0.510	0.610	0.650
18: Reasoning about size	0.531	0.520	0.620	0.950
19: Path finding	N/A	0.080	0.490	0.360
20: Reasoning about motivations	0.369	0.910	0.950	1.000
Average[6]	0.827	0.513	0.812	0.966

Table 5.2: Accuracy on the various question types in the bAbI dataset. We did not attempt question types 7 or 19, but successfully answered most of the other question types.

accuracy over the other models.

As can be seen in Table 5.2, our method achieves 90% or better accuracy on half of the question types we attempted, and has an overall average accuracy of 82.7% on the 18 question types. Our method soundly outperforms the LSTM baseline, making this the best-published result on this dataset in the weaker supervision setting that we used. Additionally, we achieve performance similar to the SVM baseline, even though the SVM has access to an additional training signal. The memory neural network, which has more representational power than our model, as well as an additional training signal, substantially outperforms our method.

We believe the performance shown in Table 5.2 adequately shows that our method for creating joint models of lexical and discourse semantics is successful. An analysis of the reasons for our errors further strengthens this point: most of the accuracy errors our method makes are due to impoverished discourse representation or candidate answer generation. For example, we do not run a coreference resolution system, so our method struggles to answer questions that require coreference; this is trivially rectified by including the output from a coreference system in the discourse graph. Our errors in question type 15 (basic deduction) are due to the passages using plural nouns in some sentences and singular nouns in others; including lemmas in the graph should improve the accuracy on these questions. For question type 20, the correct answer frequently does not appear in the passage, and so the accuracy errors are due to the candidate answer generator, not the lexical-discourse model.

The remaining errors are mainly due to our learning of just one lexical-discourse model per question type. We did this for simplicity, even if there was more than one question template, and in most cases it did not harm performance. However, in some cases, especially on question type 18 (reasoning about size), this made good models impossible to learn. In this question type there are two different wordings of the question, with opposite meaning (see the example about chocolate in Section 5.2.2). Including all of the sentences in the same lexical-discourse model clearly confused the model, as performance on that question type is on par with random chance. This would be easily fixed with a better model of sentential semantics, as we discuss in the next section.

5.2.5 Discussion

First, we note that we are not the first, nor the most successful, method for answering the reading comprehension questions in the bAbI dataset (Sukhbaatar et al., 2015; Weston et al., 2015). The memory neural networks shown in Table 5.2 are able to answer more of the question types correctly than our method is. However, this work is focused only on the question of building *lexical* models, and many of the question types require more sophisticated sentential semantics than a single lexical predicate. In addition, a nice benefit of our approach to these tasks is the ease with which background knowledge can be added to the lexical models. Using graph algorithms like PRA to reason over large knowledge bases is well understood (Gardner et al., 2015; Lao and Cohen, 2010b; Lao et al., 2011); all that is necessary to incorporate these knowledge bases into our lexical-discourse models is to connect the discourse graph to the knowledge graph. Background knowledge that takes the form of word vectors can also easily be used in PRA-style models (Gardner et al., 2013, 2014).

Second, while we used (question, answer) pairs to learn our lexical-discourse models, it should be straightforward to extend our method to instead use passages of text without associated questions. This is possible because we converted (question, answer) pairs to declarative sentences with known truth values; if we treat the sentences in a passage as true, and change words to obtain false sentences, we can similarly obtain training data for our lexical models without the need for questions and answers. This would allow for much larger datasets than can be obtained with (question, answer) pairs.

Finally, we have shown that simple graph-based methods can successfully build models of lexical semantics that incorporate discourse information. Several of the reading comprehension questions we attempted to answer, however, required richer models of sentential semantics than single lexical predicates. It should be possible to combine our lexical models with the compositional models of sentential semantics learned by state-of-the-art semantic parsers. These parsers will often first construct a *surface* parse, using the predicates found directly in the text, then try to ground these surface predicates in the language of some knowledge base. For example, Kwiatkowski et al. (2013) perform a complex search to ground a surface parse to Freebase relations and entities, while Krishnamurthy and Mitchell (2015) ground their surface predicates to a factorization-based prediction model. Our work is similar to that of Krishnamurthy and Mitchell in that we both treat lexical semantics as a prediction problem over predicate arguments; the difference is that our model incorporates discourse information, while that of Krishnamurthy and Mitchell does not. It should be straightforward to combine the methods, using our model of lexical-discourse semantics in a parser much like that of Krishnamurthy and Mitchell.

5.2.6 Conclusion

We have presented a method for combining discourse information into models of lexical semantics. This method represents the discourse as a graph, then learns models for lexical predicates that score argument pairs using features extracted from the discourse graph. We have shown the successfulness of this technique through experiments on the bAbI dataset. We have further enumerated several possible extensions to this method, describing how to easily incorporate background knowledge into the models for each predicate, and how to include these lexical semantic models into a semantic parser that models sentential semantics.

Chapter 6

Conclusion

In this thesis we have explored the issues of reasoning and reading with knowledge bases. The approach we have taken to reasoning with KBs involves extracting characteristics of a KB graph into a set of features that can be used in a machine learning model; knowledge graph characteristics are often isomorphic to the logical statements used in related reasoning techniques, which we have shown formally in this thesis. We introduced new methods for obtaining these graph characteristics that lead to a 46% increase in mean average precision over prior methods for reasoning with knowledge graphs, while simultaneously reducing running time by an order of magnitude. In addition, we have shown that these methods for reasoning with graphs can be applied more generally to models of machine reading, as they are simply methods for obtaining a feature matrix over node pairs in a graph. We demonstrated substantial improvements in recall and precision when using these methods in a standard model for relation extraction, and we showed that we can successfully incorporate background knowledge and discourse context in models of lexical semantics using these techniques.

6.1 Results Summary

While large scale knowledge bases have proven useful for a variety of tasks, they are inherently incomplete, both in the number of representable facts that are missing from the KB, and in the number of facts that are not representable in the KB at all. The ability to reason over these knowledge bases to judge the correctness of facts that are not present is thus becoming increasingly necessary. In this thesis we have presented three successive techniques for improved reasoning over very large knowledge bases (Chapter 3), based on the path ranking algorithm (Lao and Cohen, 2010b), a method that finds Horn clause rules in a graph and uses them as features in a logistic regression model.

Section 3.2 introduces a method for clustering the relations in a KB to reduce the feature sparsity of PRA. This method uses a factorization of the knowledge base tensor to obtain vector space representations of each relation in the KB, performs a clustering over this vector space, then replaces the relation IDs with a corresponding cluster ID, drammatically reducing the feature space considered by PRA. While this could potentially be done in any knowledge base, the method is most useful when there are many relations that have very similar meaning, such as in a KB populated (at least in part) with relations extracted from text.

Section 3.3 takes this idea one step further. The method in Section 3.2 replaces each (symbolic) relation ID with a (symbolic) cluster ID. Because the underlying representation is still symbolic, that method is still prone to sparsity issues (e.g., the clustering may be to fine-grained, or a relation may be on the boundary between clusters, etc.). In Section 3.3, we modify the PRA algorithm to make direct use of vector space representations for each relation. Instead of following the path types found by PRA exactly, this method allows random walks to follow edges that are *similar* to the edges in the specified path, with probability proportional to their similarity. This can be thought of as a convolution of the feature space with a vector space similarity kernel, and while it does not technically reduce the size of the feature space considered by PRA, it pushes the feature values of similar paths closer together, reducing the effective complexity of the learning problem.

The first two methods introduced in Chapter 3 involve examining global properties of a graph (i.e., a factorization of the KB tensor) to better inform a local search around query nodes. The third technique we introduce, subgraph feature extraction (Section 3.4), instead improves the efficiency and expressivity of PRA. When computing a feature matrix over node pairs in a graph, PRA must do approximately one search over the graph for each cell in the feature matrix, in order to compute a probability that is used as the value for that cell. It turns out that this probability contains very little useful information for predicting missing facts in a knowledge base; binary feature values perform just as well in prediction tasks as the probabilities do. Given this information, in Section 3.4 we introduce a simpler algorithm for generating a feature matrix over node pairs in the graph that only performs one search per row in the feature matrix, instead of one search per cell. In addition, because all of the features are binary, much more expressive features can be used. SFE improves running time over PRA by an order of magnitude, while at the same time substantially improving prediction performance.

After introducing several new techniques for reasoning over large knowledge bases, Section 3.5 formally examines the connection between these techniques and logical inference. PRA and SFE can both be thought of as performing a kind of discriminative probabilistic inference. Both methods use Horn clauses and other logic statements as their primary feature representation, but instead of making inferences using some proof-based technique, they use training data to assign weights to these Horn clauses. The Horn clauses representable by PRA consist of the set of clauses whose antecedents correspond to chains in a graph, and the modifications we introduced to PRA amount to the targeted addition of weighted disjunctions to the Horn clause. SFE expands the set of representable logic statements beyond simple Horn clauses, including some second-order statements, and can also incorporate additional features that are not easily expressed in the form of logical rules.

After an extensive discussion of reasoning with knowledge bases, Chapter 4 and Chapter 5 apply the same reasoning methods to models of reading natural language text. Chapter 4 shows that the feature matrices produced by PRA and SFE have application to more than just knowledge base completion or link prediction in a graph. We use the feature matrix as an additional factor in a model for distantly-supervised relation extraction, giving substantially better performance than prior methods for incorporating KB information into a relation extraction model. Chapter 5 demonstrates that graph-based reasoning methods can successfully model lexical semantics, in addition to the semantics of formal knowledge base relations. We show with both multiple choice science questions and simple reading comprehension questions that these techniques can learn lexical models able to reason over background information expressed in text and incorporate information from local discourse context.

The remainder of this chapter discusses the key ideas and lessons learned from this thesis, as well as opportunities for future work.

6.2 Key Ideas

6.2.1 Combining compositional and factorization techniques for KB inference

In the recent literature, there are two main approaches to the knowledge base completion task. One approach is to use some kind of factorization technique to obtain vector space representations for the entities and relations in the KB, then use those representations to predict which facts are missing. These approaches use global information about the entire graph to obtain their vector space representations, and then generally discard any other local information about the entities in the graph when making their decisions.

The other approach is to examine the local characteristics of the graph around specific entities when making a prediction. These methods, such as PRA and other random-walk-based techniques, generally do not make any use of global characteristics of the graph, instead using only information that can be found in close proximity to the query nodes in the graph.

The key idea underlying several of the contributions of this thesis is that these two approaches can be combined, using global graph characteristics to improve a local search around the query entities. The methods in Section 3.2 and Section 3.3 obtain vector space representations of the relations in the graph by performing a factorization, then use these representations to better understand local graph properties. Since the work in this thesis has been published, there has been quite a lot of other research that takes different approaches to combining local and global graph characteristics when performing knowledge base completion, but we believe we were the first to introduce this idea.

6.2.2 PRA as generating a feature matrix

PRA was introduced as an algorithm for ranking paths in a graph for the purpose of performing link prediction. A key insight that drove much of the work in this thesis was that PRA can instead be thought of as a general technique for generating feature matrices over node pairs in a graph. This is perhaps a subtle distinction, but viewing PRA in this light led directly to the work in Chapter 4, where PRA's feature matrix was included in more complex models. Thinking about the computational complexity involved in constructing PRA's feature matrix was also instrumental in the development of SFE (Section 3.4).

Using logic statements as features in a logistic regression model is not new to this thesis, nor to the path ranking algorithm; as far back as 2002, Popescul and others combined relational models with logistic regression (Popescul et al., 2002). Noting the correspondence between graph properties and logic statements is also not new to this thesis; this was clear at least as far back as Richards' and Mooney's relational pathfinding work in 1992 (Richards and Mooney, 1992). In this work, however, we have discussed a broader range of graph-based features than those used previously and shown their correspondence to logical statements. We have also introduced more scalable and expressive techniques for actually using these logic statements in feature matrices on real tasks.

Additionally, while PRA was originally tied to feature representations that are strictly isomorphic to Horn clause rules, recognizing that PRA is simply generating a feature matrix from a graph naturally suggests including as features other graph characteristics that are not as easily representable as logic statements, and we began to explore these possibilities with SFE.

6.2.3 Generating graphs from text

While the feature matrices constructed by PRA and SFE can be used with arbitrary models, their first and most straightforward use is for performing link prediction in a graph. Much of the work of this thesis, then, was in posing various problems as link prediction problems and carefully constructing a graph that contained enough information to solve the problem, but not so much that inference over the graph was intractable.

For example, Section 3.3.2 contains some discussion on constructing graphs from a large text corpus for performing knowledge base completion. We used a corpus containing a billion web pages to aid us in performing knowledge base completion; adding a node in the graph for every noun phrase in the corpus, as we did when answering reading comprehension questions in Section 5.2, would have been completely intractable. Using the ALIAS relation to connect KB entity nodes to noun phrase nodes proved crucial in making that problem solvable.

As another example, the graphs we initially constructed for answering multiple choice science questions in Section 5.1 were not connected enough, because the noun phrases used as nodes in the graph had too many modifiers. Dropping the modifiers and keeping only the head noun led to a graph that was connected enough for inference, though it lost a lot of information in the process.

It should be clear, then, that *how* the graph is constructed for a particular problem plays a big role in whether the inference is both possible and tractable. Unfortunately, the best graph structure to use is highly problem dependent, and we can only offer a few general guidelines for producing these graphs.

First, it is important to keep the average out-degree of the graph as low as possible. The space and time complexity of both PRA and SFE depend on the average out-degree of each node, and so adding edges to the graph that significantly increase the out-degree can have dramatic performance implications. There will almost always be hubs in a graph, and it is not necessarily wise to eliminate them entirely, but certain choices in graph representation can significantly increase the number of hubs in the graph. The more hubs there are, the harder inference will be.

Second, the features used by PRA and SFE generally just include sequences of edge labels, while the relation content in natural language text is frequently found in verb phrases. A naive approach to graph construction that just uses dependency parse edges in a graph will leave out the relational content of the sentences when producing PRA features. For instance, in the sentence “Horses eat hay”, the dependency path between “horses” and “hay” contains the edge labels -nsubj^{-1}-dobj-. This path, as a PRA feature, does contain some useful information, but not nearly as much as the path -“eat”-. By constructing the graph such that the relational content is on the edges of the graph, instead of in the nodes, we can make it much easier for PRA to capture useful information from the graph. Clearly a different choice of feature representation (such as the one used in Section 5.2) can compensate for different graph representations, but the important point here is that there is interplay between the tractability of graph inference, the decision of what to represent as nodes versus edges in the graph, and the specifics of the feature representation used. Care must be taken to ensure that the graph encodes information in such a way that the features extracted from it will be useful for prediction.

6.3 Future Work

6.3.1 Modeling compositional semantics

We showed in Chapter 5 that our graph-based reasoning methods can successfully model lexical semantics, at least in some circumstances. An obvious next step with this work is to use these lexical models as pieces of a larger compositional model of sentential semantics, such as a semantic parser. For example, Krishnamurthy and Mitchell (2015) recently described a semantic parser over an open predicate vocabulary using a factorization model as the underlying model of lexical semantics. This model worked well in some limited settings, but the factorization model is not capable of complex reasoning over background knowledge, nor of incorporating local discourse context. It should be straightforward to replace the factorization model of lexical semantics in Krishnamurthy's parser with our more powerful graph-based models.

6.3.2 Generating feature matrices from graphs

Chapter 3 presented a series of techniques for producing increasingly useful feature matrices over node pairs in a graph. There is still much room for improvement here. As a very simple example, we only generated feature matrices for node *pairs*; these matrices are useful for predicting missing instances of binary predicates in a knowledge base, but more general techniques are necessary for predicting instances of other kinds of predicates, either unary or n-ary. The one-sided features introduced with SFE could potentially be used for modeling unary predicates, and many of the other features could be generalized to handle node triples (or quadruples, etc.) instead of node pairs, though the search procedure to extract these features would become more complex.

There are also obviously many kinds of features that could potentially be extracted over a graph than those that we presented with SFE. While we have stressed the connection between the features used by SFE and Horn clauses, the benefit of our graph-based approach to generating features is that we need not restrict ourselves to features that a logician would think of. For instance, the out-degree at the source and target nodes would be trivial to compute and include as a feature in SFE, but very difficult to represent as a logic statement. This allows for a powerful combination of traditional logical inference with more general feature engineering, and we have only just begun to explore the possibilities in this space.

As a final point of future work, we note that we have taken PRA, an elegant model that has strong ties to logical inference, and reduced it with SFE to feature extraction over graphs. It seems clear experimentally that this is a significant improvement, but somehow doing feature engineering over graphs is not incredibly satisfying. We introduced a few kinds of features with SFE that we thought might work well, but there are many, many more kinds of features we could have experimented with (e.g., counts of paths found, path unigrams, path trigrams, conjunctions of simple paths, etc.). How should we wade our way through this mess of feature engineering? This is not a task we eagerly anticipate. One possible way around this is to turn to deep learning, whose promise has always been to push the task of feature engineering to the neural network. Some initial work on creating embeddings of graphs has been done (Bruna et al., 2013), but that work dealt with unlabeled graphs and would need significant modification to work in this setting. The recursive neural network of Neelakantan et al. (2015) is also a step in the right direction, though the spectral networks of Bruna et al. seem closer to the necessary network structure here.

6.4 Final Words

The thesis of this research has been that feature extraction from graphs is a scalable and effective means of incorporating knowledge base information into machine learning models of reasoning and reading. While we have shown substantial improvement on the relatively simple tasks of knowledge base completion, relation extraction, and modeling lexical semantics, there is a lot of work left to be done. We believe the methods and ideas presented in this work provide a solid foundation for further research in incorporating reasoning capabilities into higher-level models of machine reading.

Bibliography

Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. Dbxplorer: enabling keyword search over relational databases. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 627–627. ACM, 2002. 2.3.2

Gabor Angeli and Christopher D Manning. Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-2014)*, 2014. 2.2.4

Justin Betteridge, Alan Ritter, and Tom Mitchell. Assuming facts are expressed more than once. In *27 th International Conference of the Florida Artificial Intelligence Research Society*, 2014. 4.3

Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and Shashank Sudarshan. Keyword searching and browsing in databases using banks. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 431–440. IEEE, 2002. 2.3.2

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD*, 2008. 1, 2.1, 4.1

Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. Learning structured embeddings of knowledge bases. In *AAAI*, 2011. 2.2.4

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013. 2.2.3

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 6.3.2

J. Callan, M. Hoy, C. Yoo, and L. Zhao. Clueweb09 data set. *boston.lti.cs.cmu.edu*, 2009. (document), 3.2.2

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010. 1, 2.1, 17

Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, pages 571–580. ACM, 2007. 2.3.2

Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on*

Empirical Methods in Natural Language Processing (EMNLP), pages 1568–1579, 2014. 2.2.4

Peter Clark, Philip Harrison, and Niranjan Balasubramanian. A study of the knowledge base requirements for passing an elementary science test. In *Proceedings of the 2013 workshop on Automated knowledge base construction*, pages 37–42. ACM, 2013. 5.1, 5.1.1, 5.1.3

George R Doddington, Alexis Mitchell, Mark A Przybocki, Lance A Ramshaw, Stephanie Strassel, and Ralph M Weischedel. The automatic content extraction (ace) program-tasks, data, and evaluation. In *LREC*, 2004. 2.3.1

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014. 3.2.1

Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *IJCAI*, volume 11, pages 3–10, 2011. 1

Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011. 2.1, 2.3.1

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005. 4.4

Alberto García-Durán, Antoine Bordes, and Nicolas Usunier. Effective blending of two and three-way interactions for modeling multi-relational data. In *Machine Learning and Knowledge Discovery in Databases*, pages 434–449. Springer, 2014. 2.2.4

Matt Gardner and Tom Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2015. 3.4

Matt Gardner, Partha Talukdar, Bryan Kisiel, and Tom Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2013. 3.2, 3.3.1, 5.2.5

Matt Gardner, Partha Talukdar, Jayant Krishnamurthy, and Tom Mitchell. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2014. 3.3, 5.2.5

Matt Gardner, Partha Talukdar, and Tom Mitchell. Combining vector space embeddings with symbolic logical inference over open-domain text. In *2015 AAAI Spring Symposium Series*, 2015. 3.5, 5.1, 5.2.5

Lise Getoor and Lilyana Mihalkova. Learning statistical models from relational data. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1195–1198, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: 10.1145/1989323.1989451. URL `http://doi.acm.org/10.1145/1989323.`

1989451. 2.2.1

Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007. 2.2.1

Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *COLING*, volume 96, pages 466–471, 1996. 2.3.1

Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *Proceedings of EMNLP*, 2015. 8

Taher H Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002. 2.3.2

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics, 2011. 2.3.1, 4.1, 4.2.2

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003. 2.3.2

Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5, 2006. 2.2.3

Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. Markov logic networks for natural language question answering. *arXiv preprint arXiv:1507.03045*, 2015. 2.2.1

Jayant Krishnamurthy and Tom M Mitchell. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765. Association for Computational Linguistics, 2012. 4.1

Jayant Krishnamurthy and Tom M Mitchell. Learning a compositional semantics for freebase with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270, 2015. 5.2.5, 6.3.1

Arun Kumar, Feng Niu, and Christopher Ré. Hazy: making it easier to build and maintain big-data analytics. *Communications of the ACM*, 56(3):40–49, 2013. 3.2.2

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Citeseer, 2013. 5.2.5

Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 31–46, 2012. 3.2.3, 3.3.5

Ni Lao. *Efficient Random Walk Inference with Knowledge Bases*. PhD thesis, Carnegie Mellon University, 2012. 2.2.2, 7, 2.3.2

Ni Lao and William W Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 881–888. ACM, 2010a. 2.3.2

Ni Lao and William W Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67, 2010b. 1, 2.2.2, 3.1, 3.2.1, 3.3.4, 3.3.5, 5.2.5, 6.1

Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2011. 2.2.2, 6, 3.2.1, 5.2.5

Ni Lao, Amarnag Subramanya, Fernando Pereira, and William W Cohen. Reading the web with learned syntactic-semantic inference rules. In *Proceedings of EMNLP-CoNLL*, 2012. 3.2.1, 3.3.1, 3.3.2, 3.3.4, 3.3.5

Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995. 2.1, 4.1

Hector J Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *KR*, 2012a. 5.2.1

Hector J Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *KR*, 2012b. 4.1

Hugo Liu and Push Singh. Conceptnet: a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004. 1, 4.1, 4.2.3

Bill MacCartney and Christopher D Manning. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200. Association for Computational Linguistics, 2007. 2.2.4

Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534. Association for Computational Linguistics, 2012. 2.1, 2.3.1, 3.3.2, 5.1.2

Andrew Kachites McCallum. Mallet: A machine learning for language toolkit, 2002. URL `http://mallet.cs.umass.edu`. 3.3.5

Pablo N. Mendes, Max Jakob, and Christian Bizer. Dbpedia for nlp: A multilingual cross-domain knowledge base. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, 2012. 4.1

Einat Minkov and William W Cohen. Learning graph walk based similarity measures for parsed text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 907–916. Association for Computational Linguistics, 2008. 2.3.2

Einat Minkov, William W Cohen, and Andrew Y Ng. Contextual search and name disambiguation in email using graphs. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 27–34. ACM, 2006. 2.3.2

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011. Association for Computational Linguistics, 2009. 2.3.1

Tom M. Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapa Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In *AAAI 2015*. Association for the Advancement of Artificial Intelligence, 2015. 4.1

Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994. 2.2.1

Ndapandula Nakashole and Tom M. Mitchell. A knowledge-intensive model for prepositional phrase attachment. In *ACL 2015*. Association for Computational Linguistics, 2015. 4.2.3

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *ACL 2015*. Association for Computational Linguistics, 2015. 3.4.2, 3.4.3, 6.3.2

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816, 2011. 2.2.3

Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. In *Advances in Neural Information Processing Systems*, pages 1179–1187, 2014. 4.4

Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, 4(6):373–384, 2011. 2.2.1

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02), 2007. 3.2.2

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999. 2.3.2

Alexandrin Popescul, Lyle H Ungar, Steve Lawrence, and David M Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *Workshop on Multi-Relational Data Mining at KDD*, 2002. 6.2.2

J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990. 2.2.1, 3.5.2

Bradley L Richards and Raymond J Mooney. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 1992. 2.3.2, 6.2.2

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2): 107–136, 2006. 2.2.1

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 1, page 2, 2013. 5.2.1

Sebastian Riedel, Limin Yao, and Andrew McCallum. Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer, 2010. 4.1, 4.4

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of NAACL-HLT*, 2013. 2.2.4, 2.3.1, 3.3.2, 5

Stefan Schoenmackers, Oren Etzioni, and Daniel S Weld. Scaling textual inference to the web. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 79–88. Association for Computational Linguistics, 2008. 2.2.4, 3.5.2

Stefan Schoenmackers, Oren Etzioni, Daniel S. Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1088–1098, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1870658.1870764`. 3.5.2

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013. 2.2.4

Fabian Suchanek, James Fan, Raphael Hoffmann, Sebastian Riedel, and Partha Pratim Talukdar. Advances in automated knowledge base construction. *SIGMOD Records journal, March*, 2013. 1

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of WWW*, 2007. 2.1

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*, 2015. 5.2.5

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D Manning. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465. Association for Computational Linguistics, 2012. 2.3.1, 4.2.3

Ilya Sutskever, Joshua B Tenenbaum, and Ruslan R Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. In *Advances in neural information processing systems*, pages 1821–1828, 2009. 2.2.3

Partha Pratim Talukdar, Derry Wijaya, and Tom Mitchell. Acquiring temporal constraints between relations. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 992–1001. ACM, 2012. 3.3.2, 3.3.5, 5.1.1

Hanghang Tong, C Faloutsos, and J-Y Pan. Fast random walk with restart and its applications. In *Sixth International Conference on Data Mining (ICDM'06)*, 2006. 2.3.2

William Yang Wang, Kathryn Mazaitis, and William W. Cohen. Programming with personalized pagerank: A locally groundable first-order probabilistic logic. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Man-*

agement, CIKM '13, pages 2129–2138, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2263-8. doi: 10.1145/2505515.2505573. URL `http://doi.acm.org/10.1145/2505515.2505573`. 2.2.1, 17

William Yang Wang, Kathryn Mazaitis, and William W Cohen. Structure learning via parameter learning. *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM 2014)*, 2014a. 4

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-2014)*, 2014b. 2.3.1

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119, 2014c. 2.2.4

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *WWW*, 2014. 3.3.5

Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. Connecting language and knowledge bases with embedding models for relation extraction. In *Proceedings of EMNLP*, 2013. 2.3.1, 4.2.3

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014. 5.2.4

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: a set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015. 5.2.1, 5.2.2, 5.2.4, 5.2.5

Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007. 2.3.1

Jun Zhu. Max-margin nonparametric latent feature models for link prediction. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 719–726, 2012. 2.2.3